Indifferentiability of 3-Round Even-Mansour with Random Oracle Key Derivation

Chun Guo[1,2], and Dongdai Lin[1]

[1] State Key Laboratory of Information Security,
Institute of Information Engineering, Chinese Academy of Sciences, China
[2] University of Chinese Academy of Sciences, China
guochun@iie.ac.cn

Abstract. We revisit the t-round Even-Mansour (EM) scheme with random oracle key derivation previously considered by Andreeva et al. (CRYPTO 2013), namely,

$$\mathsf{xor}_k \circ \mathbf{P}_t \circ \mathsf{xor}_k \circ \ldots \circ \mathsf{xor}_k \circ \mathbf{P}_2 \circ \mathsf{xor}_k \circ \mathbf{P}_1 \circ \mathsf{xor}_k,$$

where $\mathbf{P}_1, \ldots, \mathbf{P}_t$ stand for t independent n-bit random permutations, xor_k is the operation of xoring with the n-bit round-key $k = \mathbf{H}(K)$ for a κ-to-n-bit bit random oracle $\mathbf{H}$ on a κ-bit main key K. For this scheme, Andreeva et al. provided an indifferentiability (from an ideal (κ, n)-cipher) proof for 5 rounds while they exhibited an attack for 2 rounds. Left open is the (in)differentiability of 3 and 4 rounds.

We present a proof for the indifferentiability of 3 rounds and thus closing the aforementioned gap. This also separates EM ciphers with non-invertible key derivations from those with invertible ones in the "full" indifferentiability setting. Prior work only established such a separation in the weaker sequential-indifferentiability setting (DCC, 2015). Our results also imply 3-round EM indifferentiable under multiple random known-keys, partially settling a problem left by Cogliati and Seurin (FSE 2016).

The key point for our indifferentiability simulator is to pre-emptively obtain some chains of ideal-cipher-queries to simulate the structures due to the related-key boomerang property in the 3-round case. The length of such chains have to be as large as the number of queries issued by the distinguisher. Thus the situation somehow resembles the context of hash-of-hash H^2 considered by Dodis et al. (CRYPTO 2012). Besides, a technical novelty of our proof is the *absence* of the so-called *distinguisher that completes all chains.*

Keywords: blockcipher, ideal cipher, indifferentiability, key-alternating cipher, iterated Even-Mansour cipher, H-coefficients technique.

Table of Contents

1 Introduction

A fundamental cryptographic problem is to construct "secure" blockciphers from permutations. A natural solution is the iterated Even-Mansour (EM) scheme (a.k.a. key-alternating cipher), which is abstracted from the widely used blockcipher design paradigm *substitution-permutation networks*. Notable instances include Rijndael – the current AES standard – and Serpent [ABK98] – the most competent contender of Rijndael. Theoretical understanding of this scheme is thereby crucial. Modeling the underlying permutations as public random permutations (RPs) and with different number of rounds, it is possible to prove (variants of) this scheme secure up to different levels such as pseudorandomness (secure in the traditional secret key setting) [EM97,BKL+12,Ste12,LPS12,CS14,CLL+14], related-key pseudorandomness (secure against related-key attacks [Bih94]) [FP15,CS15b], security in a practice-relevant multi-user setting [ML15,HT16], security against known-key attacks [ABM13,CS16], correlation intractability [CS15b,GL15c], and indifferentiability from ideal ciphers [ABD+13a,LS13,GL15a]. Although ideal models are uninstantiatable [CGH04,MRH04,Bla06], such arguments are widely accepted as showing the absence of generic attacks as well as the soundness of the design approaches.

Briefly speaking, for $\mathrm{EM}^{\mathbf{P}}$ built from RPs $\mathbf{P}$, if there exists an efficient simulator $S^{\mathbf{E}}$ that could mimic the (non-existing) underlying permutations by accessing an ideal cipher $\mathbf{E}$ (a randomly selected blockcipher), such that $(\mathbf{E}, S^{\mathbf{E}})$ looks the same as $(\mathrm{EM}^{\mathbf{P}}, \mathbf{P})$, then $\mathrm{EM}^{\mathbf{P}}$ is indifferentiable from $\mathbf{E}$ [MRH04]. This means $\mathrm{EM}^{\mathbf{P}}$ "behaves" as $\mathbf{E}$ in a well-defined sense, thus showing how to build ideal ciphers using random permutations. To establish indifferentiability, one needs to design capable simulators that could resist *all* distinguishers;[3] to disprove, one needs to exhibit a distinguisher that could collapse *all* simulators.

For different variants of EM, indifferentiability is achieved at different number of rounds. The first such results were given by Andreeva, Bogdanov, Dodis, Mennink, and Steinberger (ABDMS), on a kind of EM with strong non-invertible key derivation [ABD+13a], which we call EM with Random oracle key derivation (EMR). Formally, the t-round scheme EMR_t uses t independent n-bit RPs $\mathbf{P}_1, \ldots, \mathbf{P}_t$ and a κ-to-n-bit random oracle $\mathbf{H}$, and sets $k = \mathbf{H}(K)$,

$$\mathsf{EMR}_t.\textsc{Enc}(K, x) = k \oplus \mathbf{P}_t(k \oplus \mathbf{P}_{t-1}(\ldots \mathbf{P}_1(k \oplus x) \ldots))$$

for a key $K \in \{0,1\}^\kappa$ and a message $x \in \{0,1\}^n$. Cf. Fig. 1 (left).

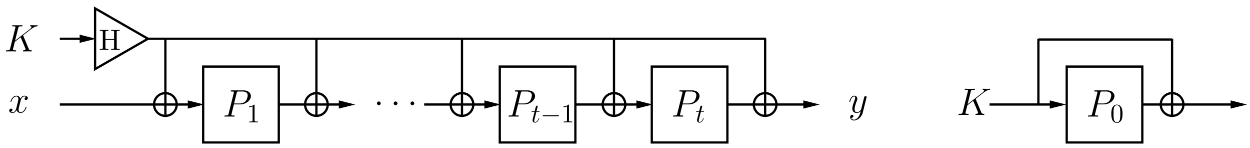


Fig. 1. (Left) the t-round EMR scheme; (Right) un-keyed Davies-Meyer key derivation $KD(K) = P(K) \oplus K$.

For such schemes ABDMS gave both positive and negative results depending on the number of rounds. For two rounds they exhibited a distinguishing attack (this negative result was indeed applicable to 2-round EM with *any key schedule*), while for five rounds they offered an indifferentiability proof. This leaves an obvious gap between the positive and the negative results, cf. Table 1. ABDMS also exhibited an attack against 3 rounds EMR_3, to show that their natural approach cannot be applied to 3 rounds. However, there's no evidence that this attack can fool *any* indifferentiability simulator. Therefore, the status of EMR_3 remains unclear for 4 years. Here we remark please do *not* take EMR_3 as *differentiable*. For this, we emphasize that in page 13 of [ABD+13a], it writes: *Firstly, no tripwire simulator with 3 rounds is secure,...* Not all simulators are "tripwire simulators".

Our Contribution. We give a positive answer on the indifferentiability of EMR_3. In the most common case, our simulator makes $O(q^4)$ queries to the ideal cipher and achieves $O(\frac{q^{12}}{N})$ security (N denotes 2^n, and will be used throughout the remaining). Although worse than ABDMS's simulator for EMR_5 (which needs $O(q^2)$ queries and delivers $O(\frac{q^{10}}{N})$ security), this does close the mentioned annoying gap, cf. Table 1. Indeed, due to the existence of pseudo-attacks[4] on EMR_3, the security lose is somewhat inevitable – although we have not been able to prove or disprove its tightness.

As ABDMS has exhibited a distinguisher on 3-round EM with (even idealized) invertible key schedule, our positive result also definitively separate such EM from EM with non-invertible key schedule in the full indifferentiability setting. Previously, such separation was only established in the weaker sequential-indifferentiability setting [GL15c] (cf. [MPS12] for sequential-indifferentiability). Optimal proofs in sequential-indifferentiability setting are much easier than their analogue in the full indifferentiability setting, and this helps [GL15c] a lot; but this also indicates our work is much more technical than [GL15c].

To reach this proof, we deeply investigate the structural properties of 3-round single-key EM. Such properties might be helpful for future research – including provable security evaluations and cryptanalytic ones. Also, this

[3] To ensure secure compositions, $\forall D \exists S$-style indifferentiability results already suffice, cf. its original definition [MRH04]. However, existing positive results are usually stronger $\exists S \forall D$-style ones, e.g. [CDMP05].

[4] Referring to the attack(s) able to fool a very large class of (but not all) simulators.

somewhat matches a conjecture of Holenstein et al.: finding optimal indifferentiability proof for Feistel requires a deep understanding of the structures. While they focused on Feistel, we think their conjecture also covers EMR.

Table 1. State of the art of indifferentiability of EMR.

Number of rounds	Indifferentiable? (by [ABD+13a])	
≤ 2	no	-
3	unclear	*yes*
4	unclear	(*trivially yes*)
≥ 5	yes	-

ABDMS also considered purely permutation-based EM variants, and suggested the most efficient solution is to use an un-keyed Davies-Meyer key derivation $KD(K) = \mathbf{P}(K) \oplus K$ to replace $\mathbf{H}$, cf. Fig. 1 (right). The RP used in this key derivation should be independent from the round-permutations. We denote this variant by EMDP_t. Our positive result on EMR_3 can be easily extended to EMDP_3, just as ABDMS extended theirs on EMR_5 to EMDP_5. This shows an ideal cipher can be built via four RP calls, which is currently the best known result.

We also observe that there exist strong relations between the indifferentiability of EMR scheme and the multiple known-key indifferentiability of SEM scheme. Concretely, our main result implies that for $\zeta > 1$, under ζ random known-keys, SEM_3 is indifferentiable from an ideal (n, n)-cipher.[5] This partially settles a problem left by Cogliati and Seurin [CS16]. As they showed SEM_2 can be attacked under 2 arbitrary known-keys, this positive result is also tight with respect to rounds.[6]

A technical contribution is to give the first analysis on idealized blockciphers without the so-called distinguisher that completes all chains (would be refereed by $\overline{D}$). In [ABD+13a], finding such a clean-cut proof was mentioned as a *significant technical innovation*; indeed, compared to the previous analysis, our proof is not much more complicated. See the technical issues below for more details.

How? The following five paragraphs sequentially list the three key steps of our indifferentiability proof for EMR_3, how did we eliminate $\overline{D}$, and how to transform the main result to indifferentiability under multiple random known-keys.

Peeling off Whitening Keys. Consider a simplified variant EMR_3^*, which is obtained by "peeling off" the two whitening keys in EMR_3:

$$\mathbf{P}_t \circ \mathsf{xor}_k \circ \ldots \circ \mathsf{xor}_k \circ \mathbf{P}_2 \circ \mathsf{xor}_k \circ \mathbf{P}_1.$$

Our first observation is that most "natural" useful simulators S for EMR_3^* could be translated into similarly useful simulators $\widetilde{S}$ for EMR_3. Thus we first focus on EMR_3^* and prove concrete bounds for it. In our opinion, this scheme-level switch simplifies the proof language as well as the illustrations a lot.

We note that a similar claim was made in [ABD+13b], the first version of [ABD+13a]: *if E is indifferentiable from* **IC** *and a, b are known constants that only depend on K (i.e. independent of the n-bit input blocks), then the cipher $\mathsf{xor}_b \circ E \circ \mathsf{xor}_a$ is also indifferentiable from* **IC**. For many FX-like cipher within imagination, the argument is clear; however, for the relation between EMR_3 and EMR_3^* the argument takes some efforts, because the whitening keys have to be derived via the same interface as the "internal" round-keys. For our simulator S for EMR_3^* we build a simulator $\widetilde{S}$ for EMR_3, and proves $\widetilde{S}$ has *exactly the same* security as S, cf. Section 12. Briefly, $\widetilde{S}$ runs S: each time S issues a query (K, z) to $\mathbf{E}$, S has simulated a derived round-key k for K, and thus $\widetilde{S}$ issues the query $(K, k \oplus z)$ to $\mathbf{E}$ and return the masked answer $k \oplus z'$ to S. The requirement of "K-simulated" is met by simulators following "natural" approaches, but not satisfied by any arbitrary effective simulator. In fact, we exhibit an artificial simulator for EMR_3^* for which our transition method and proof seem not applicable, cf. Appendix A. (This, however, does not harm our main results.)

Blocking ABDMS's Pseudo-Attack. Our second observation is based on the already mentioned ABDMS's pseudo-attack on $\mathsf{EMR}_3^*/\mathsf{EMR}_3$. We show it can be generalized to a more powerful one. Briefly speaking, ABDMS observed in EMR_3^* that if $P_1(x_{1,1}) \oplus k_1 = P_1(x'_{1,1}) \oplus k_2$, then it holds $P_1(x_{1,2}) \oplus k_1 = P_1(x'_{1,2}) \oplus k_2$ for

[5] On the other hand, for EMR_3, the "full" indifferentiability trivially implies ζ-known-key-indifferentiability.

[6] However, they conjectured 4 rounds indifferentiable under *any* set of ζ known-keys, which is not settled by us.

$x_{1,2} = \mathrm{E}^{-1}(K_2, \mathrm{E}(K_1, x_{1,1}))$ and $x'_{1,2} = \mathrm{E}^{-1}(K_1, \mathrm{E}(K_2, x'_{1,1}))$ (E and E^{-1} are the interfaces of EMR_3^*. For hints on the reason, please jump ahead for Fig. 2). Whereas we further observe that $P_1(x_{1,l}) \oplus k_1 = P_1(x'_{1,l}) \oplus k_2$ for $x_{1,l} = (\mathrm{E}_{K_2}^{-1} \circ \mathrm{E}_{K_1})^l(x_{1,1})$ and $x'_{1,l} = (\mathrm{E}_{K_1}^{-1} \circ \mathrm{E}_{K_2})^l(x'_{1,1}))$ holds for $l \geq 2$. To check this condition, the adversary mainly needs to query $\mathbf{E}/\mathsf{EMR}_3^*$ to obtain two sequences of values $x_{1,1} \xrightarrow{K_1} y_{1,1} \xrightarrow{K_2} x_{1,2} \xrightarrow{K_1} y_{1,2} \xrightarrow{K_2} \ldots$ and $x'_{1,1} \xrightarrow{K_2} y'_{1,1} \xrightarrow{K_1} x'_{1,2} \xrightarrow{K_2} y'_{1,2} \xrightarrow{K_1} \ldots$, thus it could avoid the "attention" of the simulator.

On the negative side, we conclude the natural simulation-via-chain-completion approach initiated by Coron et al. [CPS08] cannot succeed (when solely used) in this context. However, a simulator can re-gain the awareness of "what the distinguisher is trying to do", if itself prepares a structure as described, e.g. it queries $\mathbf{E}$ to get $x_{1,1} \xrightarrow{\mathrm{E}_{K_1}} y_{1,1} \xrightarrow{\mathrm{E}_{K_2}^{-1}} x_{1,2} \xrightarrow{\mathrm{E}_{K_1}} \ldots$ and $x'_{1,1} \xrightarrow{\mathrm{E}_{K_2}} y'_{1,1} \xrightarrow{\mathrm{E}_{K_1}^{-1}} x'_{1,2} \xrightarrow{\mathrm{E}_{K_2}} \ldots$, and internally sets (the simulated) $P_1(x_{1,i}) = P_1(x'_{1,i}) \oplus k_1 \oplus k_2$ and $P_3^{-1}(y_{3,i}) = P_3^{-1}(y'_{3,i}) \oplus k_1 \oplus k_2$ for each i. Intuitively, assuming the distinguisher makes at most q queries, then as long as the simulator's chains are a bit longer than q, the adversary cannot build a longer chain itself and cannot fool the simulator in such a manner. The situation is a bit similar to the hash-of-hash $H^2(M) = H(H(M))$ discussed by Dodis et al. [DRST12].

Compared to [DRST12], our work deviates in two aspects. First, EMR_3^* is a domain-extension scheme (while H^2 is not), and this significantly increase the complexity of this part. Second, when the distinguisher "moves" in the chains prepared by the simulator, Dodis et al. required the simulator to extend the chains to keep the "cursor" of the distinguisher within control. While we observe that it's not necessary, if the prepared chains are sufficiently long. To give a formal argument, we prove (via a very cumbersome analysis) that the maximum length of the chains formed by values that are "known" to the distinguisher cannot exceed the length of the chains prepared by the simulator. This technical improvement may not be very helpful for the proof for H^2, but we think it does befit us a lot, because the mechanism for our simulator to extend the structure involved in our proof (as well as the subsequent argument) would be very complicated (jumping ahead, see Fig. 3).

Our simulator distinguishes "internal" queries from distinguisher's queries to some extent. We think this partially settles a question mentioned in [DS16].

Extracting and Transferring Meta-data. To prove that the simulator can adapt with overwhelming probability is of the main sub-goals of such proofs. In general, when completing a chain C, the simulator would find the value at one side of the adaptation round obtained by a recent randomly sampling operation, e.g. x_1, the input to P_1. Clearly, w.h.p. this value is not in P_1. On the other hand, the value at the other side (e.g. y_1) may be occupied by a chain completed earlier than C. This would make the adaptation failed. To argue that this kind of event is unlikely is very cumbersome and error-prone.

To deliver our solution to this problem, let's first recall how did ABDMS succeed [ABD+13a]. Briefly speaking, ABDMS's simulator never adapt in the simulated permutations P_2 and P_4 (recall that they dealt with 5-round case). Say, they reserved P_2 and P_4 as two "random" rounds, full of "randomly answered" queries. Thus such P-queries along with the derived round-keys form many trees. Then during a recursive chain completion process, values in such trees are used (for adaptation) from the root to each leaf. Since the path between the root and each leaf is unique, no earlier chain completion could occupy the adaptation-values supposed to be used by another chain.

One may expect we could reserve P_2 as a "random" round, so that we could borrow ABDMS's argument. Unfortunately, there exists some distinguisher D that could force the simulator to adapt in P_2, cf. Appendix B.[7] Our solution to this problem is a *meta-data transferring technique*. Each time the simulator is to adapt in P_2, we find the E-query corresponding to the chain that is being completed, and associate the "meta-data" of this E-query to this "adapted 2-query". The features of these E-queries ensures some feature of the adjacent 1- and 3-queries, and this further ensure the features of the "adapted 2-queries". With the help of these features, we are finally able to prove that *all* 2-queries along with derived round-keys form directed trees, which enables the success of adaptations. Besides, this also allows us to transform a lot of intricate impossible structures into "pseudo-cycles" of 2-queries, thus enabling the arguments for impossibility. We think this technique offers a new solution to prove in extremely restricted cases – even cases with no "buffer rounds".

Go, $\overline{D}$. We first recall why indifferentiability proofs for idealized blockciphers typically rely on $\overline{D}$. For clearness, take the proof for EMR_3^* as an example, and denote by $G_2(\mathbf{E}, S)$ and $G_3(\mathsf{EMR}_3^*, \mathbf{R})$ the two systems in question

[7] Another successful line relies on "buffer rounds" or "pending queries", which are round-function-values adjacent to the adaptation-rounds, and will be defined to fresh random values *right before* adaptations [HKT11,LS13,DSKT16]. The most recent one is due to Dai and Steinberger [DS16], which also used a tree-based argument to prove the "undefinedness" of pending queries. However, in EMR_3^*, it seems we do not have enough space for such buffer rounds.

(as done in subsection 11.2). Such analysis usually proves the indistinguishability of G_2 and G_3 via a randomness mapping argument (RMA). A classical RMA would require one to specify a set of G_1 executions and a set of G_2 executions linked by a map. G_2 and G_3 executions linked by this map have the same behavior in the view of the distinguisher, and have similar probabilities of occurring.

However, note that the amount of randomness used by G_2 executions may not be the same as that used by G_3 executions. For this, assuming a distinguisher D which asks only one encryption query (K, x_1). To answer this query, in the execution $D^{G_2(\mathbf{E},S)}$, G_2 $(\mathbf{E})$ only needs to sample *1* n-bit random value. On the other hand, in $G_3(\mathsf{EMR}_3^*, \mathbf{R})$, G_3 $(\mathbf{R})$ needs to sample *4* n-bit random values k, y_1, y_2, y_3. Thus the amount of randomness needed by the two systems are different, and such two executions do not have similar probabilities of occurring – although they may have the same behaviors in the view of D.

Here lies the crucialness of $\overline{D}$: if $\overline{D}$ ensures each encyption/decryption query has their complete computation chain exist in the (G_2 or G_3) execution, then G_2 generally needs to sample *4* n-bit random values to fill in the corresponding chain, until only one round is missing; and G_2 (more clearly, S) then adapt at this round. For example, S may samples k, y_1, y_2, then obtains (the forth random value) y_3 from $\mathbf{E}$, and then adapts at P_3 (i.e. defining $P_3(y_2 \oplus k) \leftarrow y_3$). By this, the number of random values needed by two "typical" executions are the same, so that the two executions can occur with close probability.

It can be seen from the above discussion that to get rid of $\overline{D}$, the curtail point is to deal with the probability issues around the "isolated" E-queries that do not have their corresponding chains exist in the execution. In subsection 11.2 we call such "isolated" E-queries **Type II**. For such **Type II** E-queries, we only consider the probability for G_3 executions give the same answers as the G_2 executions. In other words, we ignore the "redundant" randomness used by G_3 to answer **Type II** E-queries. We finally prove that in G_3 executions, each **Type II** E-query can be associated with a unique "fresh" input-output pair (x_2, y_2) of $\mathbf{P}_2$. By this, the probabilities of G_2 and G_3 executions providing a certain tuple of answers to **Type II** E-queries are both close to $\frac{1}{N^{q_2}}$, with q_2 being the number of **Type II** E-queries. The other queries – including the **Type II** E-queries that have their corresponding chains exist – are handled with the classical RMA. In this sense, we indeed combine RMA with the H-coefficients technique [Pat09]. We call this method *partial randomness mapping argument*. In fact, to prove the indistinguishability of two random systems G_2 and G_3, the two techniques share the same core idea: they both require (either explicitly or implicitly) relating most of the G_2 and G_3 executions, such that: (i) the related G_2 and G_3 executions have the same behaviors in the view of the distinguisher; (ii) the related G_2 and G_3 executions have close probabilities of occurring. It's this shared idea that enables us to combine them.

To Indifferentiability under Multiple Random Known-keys. The relation lies in the following intuition: consider a distinguisher D against EMR_3, which first asks the random oracle to derive ζ round-keys, and the queries the permutations to figure out something. This interaction is like a known-key distinguisher D_{KK} running on SEM_3 under ζ random known-keys. Thus based on the indifferentiability simulator for EMR_3, we could build a simulator S_{KK} for this ζ random known-keys setting.

Other Related Work. On EM, besides the aforementioned security proofs, two other nice series should also be mentioned: the generic key-recovery attacks [Dae93,DKS15,NWW13,DDKS16,DDKS14], and the idea of basing tweakable blockciphers [LRW11] on EM [CLS15,Men15,CS15a].

Organization. In Section 2 we establish some convention and definitions. The formal theorem is also presented in Section 2, page 7. Then, Sections 3-5 describes our simulator as well as the underlying intuitions, while Sections 6-11 proves the main result on EMR_3^*. These are followed by Section 12, which transits the main result on EMR_3^* to EMR_3. Finally, Sections 13 and 14 transit the main result to indifferentiability of EMDP_3 and indifferentiability of SEM_3 under ζ random known-keys respectively.

2 Definitions and the Main Result

Notation for Master/Round Keys. Throughout this paper, all the master keys are denoted by the capital letter K, while all the round keys are denoted by the lower-case letter k (with superscripts or subscripts, whenever necessary). As we will see, our simulator would ensure a bijection between the main-keys and the round-keys. Thus to simply a lot of phrases "let $k_i^j = \mathbf{R}.\mathrm{H}(K_i^j)$", we strictly keep the consistency between the superscripts or subscripts of the main-keys and their corresponding round-keys, so that the superscripts or subscripts themselves are sufficient to indicated "which are whose" (and thus we omit the otherwise frequently appearing phrases "$k_i^j = \mathbf{R}.\mathrm{H}(K_i^j)$"). For example, after introducing a main-key K_i^j, we will use the notation k_i^j to refer to its round-key, and vice versa.

Ideal Primitives and Their Interfaces. A random oracle $\mathbf{H}$ is an ideal primitive which returns a random fixed-length string if x was never queried, or the same answer as before if x was previously queried. The random oracles considered in this work map κ-bit inputs to n-bit outputs. An n-bit RP $\mathbf{P}$ is a permutation that is uniformly selected from all $(N)!$ possible choices. Note that EMR_3 has access to both a random oracle and three random permutations. To simply the notation, we use the notation $\mathbf{R} = (\mathbf{H}, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$ to denote a *tuple of such random primitives*. We let such tuples provide unified interfaces, i.e. $\mathbf{R}$ provides an interface $\mathbf{R}.\mathrm{H}(K) := \{0,1\}^\kappa \to \{0,1\}^n$ for the random oracle and another interface $\mathbf{R}.\mathrm{P}(i, \delta, z) := \{1,2,3\} \times \{+,-\} \times \{0,1\}^n \to \{0,1\}^n$ for the three random permutations (i is the index, $\delta \in \{+,-\}$ indicates direct query or inverse query, and $z \in \{0,1\}^n$ is the queried value).

Ideal ciphers have been mentioned before. In the rest part, depending on the context, the notation $\mathbf{E}$ has two different meanings: in Sections 3-12, $\mathbf{E}$ refers to an ideal (κ, n)-blockcipher, and the interfaces are $\mathbf{E}.\mathrm{E}(K, z) := \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$ and $\mathbf{E}.\mathrm{E}^{-1}(K, z) := \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$; in Sections 13 and 14, $\mathbf{E}$ refers to an ideal (n, n)-blockcipher, and the interfaces are $\mathbf{E}.\mathrm{E}(K, z) := \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$ and $\mathbf{E}.\mathrm{E}^{-1}(K, z) := \{0,1\}^n \times \{0,1\}^n \to \{0,1\}^n$.

Indifferentiability. Indifferentiability framework [MRH04] addresses idealized constructions in settings where no underlying element (including building blocks and parameters) is secret. For concreteness, consider $\mathsf{EMR}_3^{\mathbf{R}}$: a distinguisher $D^{\mathsf{EMR}_3^{\mathbf{R}}, \mathbf{R}}$ with oracle access to the cascade and the underlying ideal ciphers is trying to distinguish $\mathsf{EMR}_3^{\mathbf{R}}$ from $\mathbf{E}$. Then, a formal definition due to [LS13] is as follows.

Definition 1 (Indifferentiability). *The idealized blockcipher $\mathsf{EMR}_3^{\mathbf{R}}$ with oracle access to ideal primitives $\mathbf{R}$ is said to be statistically $(q, \sigma, t, \varepsilon)$-indifferentiable from an ideal cipher $\mathbf{E}$ if there exists a simulator $S^{\mathbf{E}}$ s.t. S makes at most σ queries to $\mathbf{E}$, runs in time at most t, and for any distinguisher D which issues at most q queries, it holds*

$$Adv^{indif}_{\mathsf{EMR}_3, \mathbf{E}, S} = \left| Pr[D^{\mathsf{EMR}_3^{\mathbf{R}}, \mathbf{R}} = 1] - Pr[D^{\mathbf{E}, S^{\mathbf{E}}} = 1] \right| \leq \varepsilon$$

Such a result means that $\mathsf{EMR}_3^{\mathbf{R}}$ can safely replace $\mathbf{E}$ whenever a moderate blow-up of the adversary's time and memory requirements is acceptable, cf. [RSS11,DGHM13] for the limitations of indifferentiability. Indeed, indifferentiability has been a de-facto standard security notion beyond traditional ones such as collision intractability and indistinguishability, and has found application in various idealized constructions including hash function [CDMP05] and random permutation [HKT11].

The Main Result. Formally stated as the following theorem:

Theorem 1. *Assuming that $\mathbf{R}$ is a tuple consisting of a κ-to-n-bit random oracle and three independent random permutations. Then for the (κ, n)-blockcipher EMR_3 built from $\mathbf{R}$, there exists a simulator $\widetilde{S}$ such that*

$$Adv^{indif}_{\mathsf{EMR}_3, \mathbf{E}, \widetilde{S}} \leq \frac{2514 q_h^6 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{2359 q_e^2 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{338 q_e \cdot q_h (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{q_h^2 + q_h^4 + 8 q_e^2 + q_e \cdot q_h}{N}$$

for any distinguisher D that makes at most q_e, q_h, and q_p queries to the encryption/decryption oracle, the random oracle, and the random permutations respectively. Moreover, $\widetilde{S}$ makes at most $26 q_h \cdot (q_e + q_p) \cdot q_p^2$ queries to the ideal (κ, n)-blockcipher $\mathbf{E}$ and runs in time $O((q_e + q_p)^2 \cdot q_p^4 + q_h (q_e + q_p)^2 \cdot q_p^4)$.

The readability of Theorem 1 is a bit bad. When $q_e = q_h = q_p = O(q)$ (the most common case), the first term in $\mathrm{Adv}^{indif}_{\mathsf{EMR}_3, \mathbf{E}, \widetilde{S}}$ dominates the bound, leading it to $\mathrm{Adv}^{indif}_{\mathsf{EMR}_3, \mathbf{E}, \widetilde{S}} = O\left(\frac{q^{12}}{N}\right)$, while the two complexity bounds are $O(q^4)$ and $O(q^7)$ respectively. Thus EMR_3 is statistically $(q, O(q^4), O(q^7), O(\frac{q^{12}}{N}))$-indifferentiable from an ideal (κ, n)-cipher.

As mentioned, Theorem 1 is obtained from the following theorem on EMR_3^*, which is indeed the focus of the main body of this paper.

Theorem 2. *Assuming that* $\mathbf{R}$ *is a tuple consisting of a* κ*-to-*n*-bit random oracle and three independent random permutations. Then for the* (κ, n)*-blockcipher* EMR_3^* *built from* $\mathbf{R}$*, there exists a simulator* S *such that*

$$Adv^{indif}_{\mathsf{EMR}_3^*,\mathbf{E},S} \leq \frac{2514 q_h^6 (q_e+q_p)^2 \cdot q_p^4}{N} + \frac{2359 q_e^2 (q_e+q_p)^2 \cdot q_p^4}{N} + \frac{338 q_e \cdot q_h (q_e+q_p)^2 \cdot q_p^4}{N} + \frac{q_h^2 + q_h^4 + 8q_e^2 + q_e \cdot q_h}{N}$$

for any distinguisher D *that makes at most* q_e*,* q_h*, and* q_p *queries to the encryption/decryption oracle, the random oracle, and the random permutations respectively. Moreover,* S *makes at most* $26 q_h \cdot (q_e+q_p) \cdot q_p^2$ *queries to the ideal* (κ, n)*-blockcipher* $\mathbf{E}$ *and runs in time* $O((q_e+q_p)^2 \cdot q_p^4 + q_h (q_e+q_p)^2 \cdot q_p^4)$*.*

Like [DRST12], our simulator S must know ahead the maximum number of queries the distinguisher is to make, but does not otherwise depend on its concrete distinguishing strategy. Thus similarly to [HKT11], the result proved in this paper implies EMR_3^* indifferentiable under the original definition of Maurer et al. [MRH04], but not under the stronger one of Coron et al. [CDMP05].

3 Naïve Tripwire Simulator for EMR_3^*

3.1 Explicit Randomness

Randomly sampled answers are indispensable for the simulator S to simulate RO and RPs. To handily describe how such random answers are drawn, we follow [CS15b] and make the randomness explicit via a tuple of random primitives $\mathbf{R} = (\mathbf{H}, \mathbf{P}_1, \mathbf{P}_2, \mathbf{P}_3)$. We denote by $S^{\mathbf{E},\mathbf{R}}$ the simulator which accesses $\mathbf{E}$ and $\mathbf{R}$. Using such explicit randomness has no difference with lazily sampling at the beginning of the experiment [ABD+13a,CS15b].

3.2 Maintaining History

We use the notation $\textsc{Or}(z) \to z'$ to indicate that the oracle-interface $\textsc{Or}$ returns z' when queried at z.

To maintain the query-history, we take the explicit bookkeeping technique introduced by ABDMS [ABD+13a], i.e. keeping the "meta-data" besides the query-answer pairs themselves. More clearly, our simulator S maintain a set $Queries$ storing tuples of the form (i, x_i, y_i, dir, num), where $i \in \{1, 2, 3\}$ records the index of the (simulated) underlying permutation, $x_i, y_i \in \{0,1\}^n$ record the query and simulated answer, while the value $dir \in \{\to, \leftarrow, \perp\}$ indicates the direction of the query: $dir = \to$ indicates forward query, $dir = \leftarrow$ indicates inverse query, while $dir = \perp$ indicates the "adapted" queries (AD-queries for short), i.e. queries internally defined for consistency of chains. The field num is the value of a query counter $qnum$ (initialized to 1 at the beginning of the interaction) when this record is created. The last coordinate (num) or last two coordinates (including dir) are often omitted, when they are not of interest to the discussion at hand. Such a tuple (i, x_i, y_i) is thus called an *i-query*; and the terminology *P-query* refers to any i-query indifferently to the value of i. Whenever S newly simulates a query (i, x_i, y_i) (as the result of either answering D's query or S's inner actions), we say it *creates a new i-query*, and a record as described is added to $Queries$.

For queries to H, S maintain another set $HQueries$ storing tuples of the form (K, k, num), where $K \in \{0,1\}^\kappa$ and $k \in \{0,1\}^n$ record the queried main-key and the derived round-key respectively, while num is the value of a query counter $qnum$ (as described) when this record is created. Such a tuple (K, k) is called an *H-query*.

S would ensure that for each tuple (i, x_i), there is *at most one* y_i such that $(i, x_i, y_i) \in Queries$, and vice versa; whenever the situation forces S to break such consistency, S *aborts*. By this, the set $Queries$ is expected to define three partial permutations for $i = 1, 2, 3$, and we write P_i and P_i^{-1} for the sets $\{x_i : \exists y_i \text{ s.t. } (i, x_i, y_i) \in Queries\}$ and $\{y_i : \exists x_i \text{ s.t. } (i, x_i, y_i) \in Queries\}$ respectively, and write $P_i(x_i)$ ($P_i^{-1}(y_i)$, resp.) for the (unique) corresponding y_i (x_i, resp.) when $x_i \in P_i$ ($y_i \in P_i^{-1}$, resp.).

We note that the records in $HQueries$ defines a map between some main-keys K and round-keys k. S would also ensure the map to be bijective (if not possible then it aborts). Thus we write $HTable$ for the domain of this map and $HTable(K)$ for the image k, and $\mathcal{Z}$ for the codomain of this map.[8] Note that by the above mechanism, S *never overwrites anything*. Moreover, the sets P_i, P_i^{-1}, $HTable$, etc. change as new records are added to $Queries$ and $HQueries$.

Usually, the simulator has to pre-emptively complete some computation chains to ensure the consistency of simulated answers. For this, we call a triple (K, x_1, y_3) such that a query $\mathrm{E}(K, x_1) \to y_3$ or $\mathrm{E}^{-1}(K, y_3) \to x_1$ has appeared an *E-query*, and call a 5-tuple of queries

$$(K, k), (K, x_1, y_3), (1, x_1, y_1), (2, x_2, y_2), (3, x_3, y_3)$$

[8] I.e. $HTable$ is $\{K : \exists k, num \text{ s.t. } (K, k, num) \in HQueries\}$, and $\mathcal{Z}$ is $\{k : \exists K, num \text{ s.t. } (K, k, num) \in HQueries\}$.

with $y_1 \oplus x_2 = y_2 \oplus x_3 = k$ a *K-completed path*. Such a path indicates a *cycle* $x_1 - y_1 - x_2 - y_2 - x_3 - y_3 - (x_1)$. To avoid re-completing the same path, we let S maintain a set *Completed* to keep track of them. Once a path $(K, k), (K, x_1, y_3), (1, x_1, y_1), (2, x_2, y_2), (3, x_3, y_3)$ is completed, i.e. all the five queries have been in the history, we let S add three triples $(1, K, x_1)$, $(2, K, x_2)$, and $(3, K, x_3)$ to *Completed*. Adding three triples might be a bit redundant, but this simplifies the design.

3.3 Chain Detection: Tripwires

We first exhibit an example for the simulator detecting a chain. If D has asked $\mathrm{H}(K) \to k$ and $\mathrm{P1}(x_1) \to y_1$ and $\mathrm{P2}(x_2) \to y_2$ for $x_2 = k \oplus y_1$, then as the 3-query is the only missing one of the chain, D knows $\mathrm{P3}(x_3) = \mathrm{E}(K, x_1)$ ($x_3 = k \oplus y_2$) if it's in the real world. In this case, it seems better if the simulator is also aware of the last relation. Indeed, the *naïve tripwire simulator* of ABDMS would detect a chain $x_1 - y_1 - x_2$ upon D querying $\mathrm{P2}(x_2)$, and then complete the chain to enforce the relation $\mathrm{P3}(x_3) = \mathrm{E}(K, x_1)$ – for example, the simulator may first get $y_2 \leftarrow \mathbf{R}.\mathrm{P2}(x_2)$ and then creates an AD-3-query $(3, x_3, y_3)$ with $x_3 = y_2 \oplus k$. ABDMS called this detection condition a *12-tripwire*; to save some letters, we abbreviate it as *12-TP*.

Similarly, we could design three additional TPs, i.e. *23-*, *21-*, and *32-TPs*. We could also let it "penetrate" $\mathbf{E}$, say, design *13-* and *31-TPs*. These constitute all the TPs of the naïve simulator for EMR_3 [ABD+13a]. They are summarized as follows (in each of the following cases, the simulator detects chains).

- *12-TP*: (as mentioned) upon $\mathrm{P2}(x_2)$, if there exits $k \in \mathcal{Z}$ and $(1, x_1, y_1)$ with $y_1 \oplus k = x_2$;
- *32-TP*: similar to *12-TP* by symmetry;
- *21-TP*: upon $\mathrm{P1}^{-1}(y_1)$, if there exits $k \in \mathcal{Z}$ and $(2, x_2, y_2)$ with $y_1 = x_2 \oplus k$;
- *23-TP*: similar to *21-TP* by symmetry. Throughout the remaining we say *MidTP* to indifferently refer to 21- and 23-TP, since they are "formed" at the middle of the construction;
- *13-TP*: upon $\mathrm{P1}(x_1)$, if there exits $(K, k) \in \mathit{HQueries}$ and $(3, x_3, y_3) \in \mathit{Queries}$ such that $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$;
- *31-TP*: similar to *13-TP* by symmetry.

Note that when D querying $\mathrm{H}(K) \to k$, if there exist (i, x_i, y_i) and $(i+1, x_{i+1}, y_{i+1})$ such that $k = y_i \oplus x_{i+1}$, then new partial chains are also formed. However, if k is a random round-key newly given by $\mathbf{R}.\mathrm{H}$, then $k = y_i \oplus x_{i+1}$ is unlikely. Therefore, in the context of EMR_3, the possibility of new partial chains formed due to D querying H can be ignored, and the above six TPs constitute all the chain-detection conditions of the naïve tripwire simulator.

However, for EMR_3^* which has two less whitening keys, we have to consider an additional case: imagine D has asked $\mathrm{E}(K, x_1) \to y_3$ and $\mathrm{P3}^{-1}(y_3) \to x_3$, and then asks $\mathrm{P1}(x_1)$. After the simulator gives the answer y_1, it seems like that a 2-query is the only missing query of the chain $x_3 - y_3 - -x_1 - y_1$, and the simulator should detect a chain – somewhat like the requirement of a *13-TP*. However, note that D *did not* query $\mathrm{H}(K)$, thus the simulator does not know for which K it should check $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$.

Our solution to this case is straightforward: as long as D does not query $\mathrm{H}(K)$, the partial-chain $x_3 - y_3 - -x_1 - y_1$ is harmless; on the other hand, since $\mathrm{P1}(x_1) \to y_1$ is queried, once D queries $\mathrm{H}(K) \to k$, the simulator "knows" the key K, and is able to detect the partial-chain $y_2 - x_3 - y_3 - -x_1 - y_1 - x_2$ ($x_2 = k \oplus y_1$ and $y_2 = k \oplus x_3$) by checking whether $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$ and create an AD-2-query $(2, x_2, y_2, \perp)$ to complete it. Thus the naïve simulator for EMR_3^* contains the following additional detection condition besides the six mentioned ones:

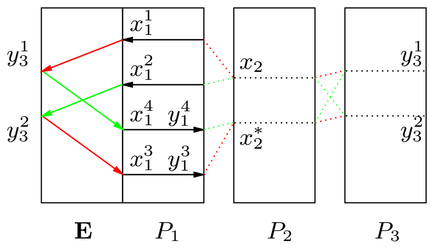
- *H-TP*: upon $\mathrm{H}(K)$, if there exits $(1, x_1, y_1), (3, x_3, y_3) \in \mathit{Queries}$ such that $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$. We re-stress that *this condition is usually not necessary in "natural" simulator for EMR_3.*

But as shown by ABDMS, the naïve simulator can be attacked. Recalling this attack is the duty of the next section.

4 Extending ABDMS's Pseudo-Attack: Motivating the Rhizome Strategy

4.1 Attack on the Naïve Simulator

With the chain-detection conditions in mind (described in the previous section), the distinguisher D of ABDMS chooses $x_2 \in \{0,1\}^n$, $K_1, K_2 \in \{0,1\}^\kappa$, $K_1 \neq K_2$, and queries $\mathrm{H}(K_1) \to k_1$, $\mathrm{H}(K_2) \to k_2$. Then D queries $\mathrm{P1}^{-1}(x_2 \oplus k_1) \to x_1^1$, $\mathrm{P1}^{-1}(x_2 \oplus k_2) \to x_1^2$, $\mathrm{E}(K_1, x_1^1) \to y_3^1$, $\mathrm{E}(K_2, x_1^2) \to y_3^2$, $\mathrm{E}^{-1}(K_2, y_3^1) \to x_1^4$, $\mathrm{E}^{-1}(K_1, y_3^2) \to x_1^3$, $\mathrm{P1}(x_1^3) \to y_1^3$, $\mathrm{P1}(x_4) \to y_1^4$. Interacting with EMR_3^*, $y_1^3 \oplus k_1 = y_1^4 \oplus k_2$ always holds (see Fig. 2 (left)). But according to the detecting conditioned mentioned before, the simulator is "bypassed" by D, and does nothing more than randomly sampling answers, and thus failed.



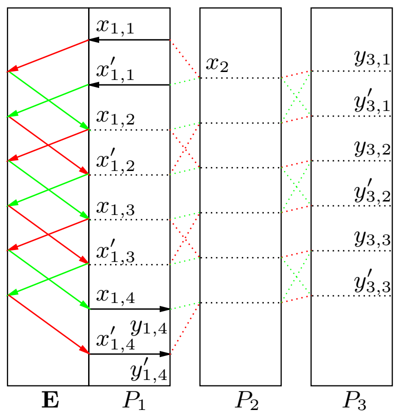


Fig. 2. (Left) The related-key boomerang structure used by ABDMS. (Right) Extending the boomerang structure: the case of $l = 3$. In each figure, the left-most rectangle denotes the "encryption area". Points x_1 and y_3 joined by a red (green, resp.) solid line satisfy the relation $\mathrm{E}(K_1, x_1) = y_3$ ($\mathrm{E}(K_2, x_1) = y_3$). The other three rectangles denote the three permutations, with black lines indicating input-output pairs. Points y_i and x_{i+1} joined by a red (green, resp.) dotted line satisfy the relation $x_{i+1} = y_i \oplus k_1$ ($x_{i+1} = y_i \oplus k_2$, resp.). Finally, the solid lines are the queries that really appear in the attacks – moreover the arrows of the directed lines indicate the query-directions of the distinguisher.

4.2 An Extended Attack

In fact, the above attack is not that powerful. If the simulator notices (by querying **E**) the fact $\mathrm{E}(K_1, x_1^1) = y_3^1 \wedge \mathrm{E}^{-1}(K_2, y_3^1) = x_1^4$ upon receiving $\mathrm{P1}(x_1^3)$, then it could go one step ahead and block the above attack. Indeed, several months before, we tried to dig out a proof with this simple idea. But then we notice the attack can be extended. Briefly speaking, the two chains of E-queries $x_1^1 - y_3^1 - x_1^4$ and $x_1^2 - y_3^2 - x_1^3$ involved in ABDMS's attack are both of length 2; if we extend them, similar properties still hold.

(1) Chooses $x_2 \in \{0,1\}^n$, $K_1, K_2 \in \{0,1\}^\kappa$, $K_1 \neq K_2$, and queries $\mathrm{H}(K_1) \to k_1$, $\mathrm{H}(K_2) \to k_2$, $\mathrm{P1}^{-1}(x_2 \oplus k_1) \to x_{1,1}$, and $\mathrm{P1}^{-1}(x_2 \oplus k_2) \to x'_{1,1}$;

(2) For $t = 2l$, makes $2 \cdot t$ queries to E and E^{-1}: $x_{1,1} \xrightarrow{\mathrm{E}_{K_1}} y_{3,1} \xrightarrow{\mathrm{E}_{K_2}^{-1}} x_{1,2} \xrightarrow{\mathrm{E}_{K_1}} y_{3,2} \xrightarrow{\mathrm{E}_{K_2}^{-1}} x_{1,3} \xrightarrow{\mathrm{E}_{K_1}} \ldots \xrightarrow{\mathrm{E}_{K_2}^{-1}} x_{1,l+1}$, $x'_{1,1} \xrightarrow{\mathrm{E}_{K_2}} y'_{3,1} \xrightarrow{\mathrm{E}_{K_1}^{-1}} x'_{1,2} \xrightarrow{\mathrm{E}_{K_2}} y'_{3,2} \xrightarrow{\mathrm{E}_{K_1}^{-1}} x'_{1,3} \xrightarrow{\mathrm{E}_{K_2}} \ldots \xrightarrow{\mathrm{E}_{K_1}^{-1}} x'_{1,l+1}$;

(3) $\mathrm{P1}(x_{1,l+1}) \to y_{1,l+1}$, $\mathrm{P1}(x'_{1,l+1}) \to y'_{1,l+1}$.

Interacting with EMR_3^*, $y_{1,l+1} \oplus k_1 = y'_{1,l+1} \oplus k_2$ always holds. This can be seen by naturally extending the (smaller) structure utilized by ABDMS, e.g. see Fig. 2 (right) for the case of $l = 3$. In fact, it's not hard to see the parameter t does not necessarily need to be even, as $\mathrm{P3}^{-1}(y_{3,i}) \oplus k_1 = \mathrm{P3}^{-1}(y'_{3,i}) \oplus k_2$ holds for each involved pair $(y_{3,i}, y'_{3,i})$.

This extended attack ruins out the hope of patching the naïve tripwire simulator with some simple tweaks. By this attack, it seems like that a capable simulator has to compute the two chains of E-queries before D computing them, and prepare for adaptations around the two E-query-chains. Although such chains can be infinitely long, D only issues a limited number of queries. Therefore, to prevent D querying in some "unready" zone, it is already enough for the simulator to prepare chains with polynomial length. The case is thus similar to the context of the hash-of-hash $H^2(M) = H(H(M))$ analyzed by Dodis et al. [DRST12]. Borrowing their ideas, we design the *rhizome strategy* for the simulator. See the next subsection for details.

4.3 Rhizome Simulation Strategy

We introduce some notions first. Two E-queries are *adjacent* if they share the same x_1 or y_3 value. We call the query structure consisting of a sequence of adjacent E-queries $(K_1, x_{1,1}, y_{3,1}), (K_2, x_{1,2}, y_{3,1}), (K_3, x_{1,2}, y_{3,2}), \ldots$ an *E-chain*, informally written as $x_{1,1} \frac{K_1}{} y_{3,1} \frac{K_2}{} x_{1,2} \frac{K_3}{} y_{3,2} - \ldots$, with the number of involved E-queries being its *length*. In such a chain, if two different keys K_1 and K_2 appear alternatively, then it's a (K_1, K_2)-*alternated E-chain*, e.g. $(K_1, x_{1,1}, y_{3,1}), (K_2, x_{1,2}, y_{3,1}), (K_1, x_{1,2}, y_{3,2}), \ldots$

From the extended attack, we conclude that a capable simulator S should prepare a structure similar to that in Fig. 2 (right) to fool the distinguisher in future. Note that in the attacks above, although no TP is set off, S has derived two round-keys k_1 and k_2, and has received two 1-queries $(1, x_{1,1}, y_{1,1})$ and $(1, x'_{1,1}, y'_{1,1})$ with $y_{1,1} \oplus y'_{1,1} = k_1 \oplus k_2$. It's now sufficient for S to uniquely determine the two (K_1, K_2)-alternated E-chains appeared in Fig. 2 (right) (i.e. the two chains starting from $x_{1,1}$ and $x'_{1,1}$ respectively). Therefore, the chain-detection condition for this mechanism is D *querying* $\mathrm{P1}^{-1}(y_1)$ as well as D *querying* $\mathrm{P3}(x_3)$, the symmetrical case.

Assuming D makes at most q_e, q_h, and q_p queries to $\mathrm{E}/\mathrm{E}^{-1}$, H, and $\mathrm{P}i/\mathrm{P}i^{-1}$ respectively. Intuitively, D querying H would not be helpful for it to "move" out from the structure prepared by S, thus it's enough for S to prepare two (K_1, K_2)-alternated E-chains with length longer than $q_e + q_h$. Our choice is to let the length be $2t$, with $2t = q_e + q_p + 3$ when $q_e + q_p$ is odd, and $2t = q_e + q_p + 4$ otherwise. Thus S should query $y_{3,1+i} \leftarrow (\mathrm{E}_{K_1} \circ \mathrm{E}^{-1}_{K_2})^i(\mathrm{E}(K_1, x_{1,1}))$, $x_{1,1+i} \leftarrow (\mathrm{E}^{-1}_{K_2} \circ \mathrm{E}_{K_1})^i(x_{1,1})$, $y'_{3,1+i} \leftarrow (\mathrm{E}_{K_1} \circ \mathrm{E}^{-1}_{K_2})^i(\mathrm{E}(K_2, x'_{1,1}))$, and $x'_{1,1+i} \leftarrow (\mathrm{E}^{-1}_{K_1} \circ \mathrm{E}_{K_2})^i(x'_{1,1})$ for i from 1 to t. Then, S could internally set $P_1(x_{1,i+1}) \oplus P_1(x'_{1,i+1}) = k_1 \oplus k_2$ and $P_3^{-1}(y_{3,i}) \oplus P_3^{-1}(y'_{3,i}) = k_1 \oplus k_2$ for each involved pairs $(x_{1,i}, x'_{1,i})$ and $(y_{3,i+1}, y'_{3,i+1})$.

The above process is somewhat like S extending a rhizome underground (two alternated E-chains, in the query history of $\mathbf{E}$) and then making a series of structures of the form $x_{1,i} - y_{1,i} - y_{1,i} \oplus k_1 \oplus k_2 - y'_{1,i} - x'_{1,i}$ "grow out of the ground". Thus we name it *rhizome strategy*, and call the structures of the form $x_{1,i} - y_{1,i} - y_{1,i} \oplus k_1 \oplus k_2 - y'_{1,i} - x'_{1,i}$ ($y_{3,i} - x_{3,i} - x_{3,i} \oplus k_1 \oplus k_2 - x'_{3,i} - y'_{3,i}$, resp) *11-shoot* (*33-shoot*, resp).

However, for the same pair of keys (K_1, K_2), D could switch the order of their applications and construct a structure symmetrical to Fig. 2 (right). By this, S has to prepare two additional (K_1, K_2)-alternated E-chains: $y_{3,1-i} \leftarrow (\mathrm{E}_{K_2} \circ \mathrm{E}^{-1}_{K_1})^i(\mathrm{E}(K_2, x_{1,1}))$, $x_{1,1-i} \leftarrow (\mathrm{E}^{-1}_{K_1} \circ \mathrm{E}_{K_2})^i(x_{1,1})$, $y'_{3,1-i} \leftarrow (\mathrm{E}_{K_2} \circ \mathrm{E}^{-1}_{K_1})^i(\mathrm{E}(K_1, x'_{1,1}))$, and $x'_{1,1-i} \leftarrow (\mathrm{E}^{-1}_{K_2} \circ \mathrm{E}_{K_1})^i(x'_{1,1})$ for i from 1 to t. To avoid negative integers in subscripts, we let S rename the mentioned $(x_{1,1}, x'_{1,1})$ as $(x_{1,t+1}, x'_{1,t+1})$, and then evaluate to the two "sides" till obtaining $x_{1,1}$ and $x_{1,2t+1}$.

4.4 Procedure ProcessShoot

In our pseudocode, the above mechanism is implemented by a procedure Process11Shoot. More clearly, upon D querying $\mathrm{P1}^{-1}(y_1) \to x_1$, if S detects $(1, x'_1, y'_1)$ with $y_1 \oplus y'_1 = k_1 \oplus k_2$, then it keeps a record of it,[9] which finally leads to making a call to Process11Shoot(x_1, y_1, K_1, K_2) to "process" this 11-shoot. This call takes (x_1, x'_1) as $(x_{1,t+1}, x'_{1,t+1})$, and has four steps:

(1) *Make-E-Chain-Phase*: makes $4t$ pairs of queries to $\mathbf{E}$ to form two (K_1, K_2)-alternated E-chains with length $4t$. They are adjacent to $x_{1,t+1}$ and $x'_{1,t+1}$ respectively, as depicted in Fig. 3 (top left):

$$x'_{1,1} \xleftarrow{\mathrm{E}^{-1}_{K_2}} \ldots \xleftarrow{\mathrm{E}_{K_1}} x'_{1,t} \xleftarrow{\mathrm{E}^{-1}_{K_2}} y'_{3,t} \xleftarrow{\mathrm{E}_{K_1}} x'_{1,t+1} \xrightarrow{\mathrm{E}_{K_2}} y'_{3,t+1} \xrightarrow{\mathrm{E}^{-1}_{K_1}} x'_{1,t+2} \xrightarrow{\mathrm{E}_{K_2}} \ldots \xrightarrow{\mathrm{E}^{-1}_{K_1}} x'_{1,2t+1},$$

$$x_{1,1} \xleftarrow{\mathrm{E}^{-1}_{K_1}} \ldots \xleftarrow{\mathrm{E}_{K_2}} x_{1,t} \xleftarrow{\mathrm{E}^{-1}_{K_1}} y_{3,t} \xleftarrow{\mathrm{E}_{K_2}} x_{1,t+1} \xrightarrow{\mathrm{E}_{K_1}} y_{3,t+1} \xrightarrow{\mathrm{E}^{-1}_{K_2}} x_{1,t+2} \xrightarrow{\mathrm{E}_{K_1}} \ldots \xrightarrow{\mathrm{E}^{-1}_{K_2}} x_{1,2t+1}.$$

In the following sections, we will call the chain adjacent to $x'_{1,t+1}$ the *old E-chain* of the Process11Shoots-call, while the chain adjacent to (the newer) $x_{1,t+1}$ the *new E-chain*.

(2) *Shoot-Growing-Phase*: ensure that each node in the old E-chain is attached by a 1- or 3-query correspondingly. For example, for $x'_{1,i}$, if $x'_{1,i} \notin P_1$, then it creates $(1, x'_{1,i}, y'_{1,i}, \to)$, cf. Fig. 3 (top right). If this new query sets off 31-TP, then pauses to create AD-2-queries to complete them – for example, if for $K' \in HTable \backslash \{K_1, K_2\}$ and $(3, \overline{x_3}, \overline{y_3})$ it holds $\mathbf{E}.\mathrm{E}(K', x'_{1,i}) = \overline{y_3}$, then it creates $(2, y'_{1,i} \oplus k', \overline{x_3} \oplus k', \perp)$. Note that K_1 and K_2 are excluded in this checking-process because the temporary 13-/31-TP parameterized by them will be settled in the next *Fill-in-Rung-Phase*. Also note that if this phase is completed as expected, then except for the 11-shoot formed by $(1, x_{1,t+1}, y_{1,t+1})$ and $(1, x'_{1,t+1}, y'_{1,t+1})$, all the other queries $(1, x'_{1,i}, y'_{1,i})$ and $(3, x'_{3,i}, y'_{3,i})$ do not form shoot parameterized by k_1 and k_2. We use "incomplete shoots" to refer to the structure formed by these queries.

(3) *Fill-in-Rung-Phase*: for each E-query in the old E-chain, if the path corresponding to it has not been complete, then create an AD-2-query to complete this path. This process is somewhat like using AD-2-queries as "rungs" to fill in a "ladder structure", cf. Fig. 3 (bottom left). Note that in this phase, Process11Shoot takes the old E-chain as a tree rooted at $x'_{1,t+1}$, and creates the corresponding AD-2-queries in a "top-down" manner in this tree, cf. the numbers in Fig. 3 (bottom left).

[9] For how to keep this record, please jump ahead to subsection 5.1. This subsection focuses on the flow of ProcessShoot.

(4) *Shoot-Completing-Phase*: for each "incomplete shoot" created in the *Shoot-Growing-Phase*, complete the by filling in a proper AD-1- or AD-3-query. For example, for $(1, x'_{1,i}, y'_{1,i})$, let $y_{1,i} \leftarrow y'_{1,i} \oplus k_1 \oplus k_2$ and create $(1, x_{1,i}, y_{1,i}, \perp)$, cf. Fig. 3 (bottom right). In this phase, PROCESS11SHOOT takes the new E-chain as a tree rooted at $x_{1,t+1}$, and creates the corresponding AD-1-queries in a "top-down" manner in this tree (similarly to the *Fill-in-Rung-Phase*), cf. the numbers in Fig. 3 (bottom right).
Here we believe the order of adaptations is not crucial. However, it seems that the argument is easier to made for this "top-down" order. For this one could jump ahead to see the proof of Proposition 21.

Upon D querying $\mathrm{P3}(x_3) \rightarrow y_3$, if S detects $(3, x'_3, y'_3)$ with $x_3 \oplus x'_3 = k_1 \oplus k_2$, then it finally calls PROCESS33SHOOT(x_3, y_3, K_1, K_2) to process this 33-shoot. The flow of this call is similar to PROCESS11SHOOT, leading to a structure symmetrical to the latter one. We similarly identify the old and the new E-chains for the PROCESS33SHOOT-call. Throughout the remaining, we would use PROCESSSHOOT procedure/call to indifferently refer to PROCESS11SHOOT or PROCESS33SHOOT, and speak G_2 *processes a 11-/33-shoot* to refer to the above processes. It's not hard to see that once enhanced with this mechanism, the naïve simulator cannot be fooled by ABDMS's attack nor our extended version any more.

4.5 Going Beyond Two Keys

The above only exhibited the simplest instance. In fact, D could force S to simultaneously detect several shoots that share some P-queries; as a consequence, their corresponding structures would interfere each other. For example, D may first query $\mathrm{H}(K_1) \rightarrow k_1$, $\mathrm{H}(K_2) \rightarrow k_2$, $\mathrm{H}(K_3) \rightarrow k_3$, then chooses a value $y_1 \in \{0,1\}^n$ and queries $\mathrm{P1}^{-1}(y_1) \rightarrow x_1$, $\mathrm{P1}^{-1}(y'_1) \rightarrow x'_1$ where $y'_1 = y_1 \oplus k_1 \oplus k_2$, $\mathrm{P1}^{-1}(y''_1) \rightarrow x''_1$ where $y''_1 = y_1 \oplus k_1 \oplus k_3$. Upon the last query $\mathrm{P1}^{-1}(y''_1)$, S detects two newly formed shoots $y_1 - x_1 - y_1 \oplus k_1 \oplus k_3 - y''_1 - x''_1$ and $y'_1 - x'_1 - y'_1 \oplus k_2 \oplus k_3 - y''_1 - x''_1$, and they share the 1-query $(1, x''_1, y''_1)$. One could see that the two corresponding relate-key boomerang structures indeed share the following completed path, cf. Fig. 4 (left):

$$(K_3, k_3), (K_3, x''_1, y''_3), (1, x''_1, y''_1), (2, x''_2, y''_2), (3, x''_3, y''_3),\ k_3 = y''_1 \oplus x''_2 = y''_2 \oplus x''_3.$$

But fortunately, we are able to prove that the share queries/interference between different shoots are limited, while the structure shared by them is consistent in the context of EMR_3^*. Thus our mechanism is able to handle such complicated case(s).

On the other hand, for a fixed pair of 1-queries $(1, x_1, y_1)$ and $(1, x'_1, y'_1)$ and four distinct keys $k_1, k_2, k_3, k_4 \in \mathcal{Z}$, if both $y_1 \oplus y'_1 = k_1 \oplus k_2$ and $y_1 \oplus y'_1 = k_3 \oplus k_4$, then $k_1 \oplus k_2 \oplus k_3 \oplus k_4 = 0$. In this case, in the call PROCESS11SHOOT(x_1, y_1, K_1, K_2), each time S completes a new shoot structure (under k_1 and k_2), it would detect a new shoot under k_3 and k_4, and has to deal with it (and vice versa), cf. Fig. 4 (right). This would make the situation extremely complicated. However, note that round-keys in $\mathcal{Z}$ are all derived by $\mathbf{R}$.H, thus it's unlikely that four round-keys $k_1 \oplus k_2 \oplus k_3 \oplus k_4 = 0$ exists. By this, it can be seen that w.h.p. *a fixed pair of 1-queries can form at most one 11-shoot.* Similar claim holds for 3-queries.

As we have introduced the most sophisticated mechanism, we will complete the design of S in the next section.

5 Completing the Design of the Simulator for EMR_3^*

Our final design inherits all the TPs from the naïve simulator as well as incorporates the aforementioned rhizome strategy. Next, subsection 5.1 first summarizes how S handles D's queries, and then subsection 5.2 gives a formal description in pseudocode.

5.1 Handling New Queries

We first remark that the AD-queries internally created by S also set off TPs and shoots, once they meet the constraints. Thus S may have to recursively complete a lot of chains; following [ABD+13a], we call such a process a *chain-reaction.* However, in cases of D querying P1, P2, $\mathrm{P2}^{-1}$, $\mathrm{P3}^{-1}$, and H, w.h.p. the possible newly-created AD-queries would not set off new TPs, thus the chain-completion is a "one-shoot-deal". Thus we first make discussion on these simpler cases. The complicated recursive chain-completion process only occurs when D queries $\mathrm{P1}^{-1}(y_1)$ and $\mathrm{P3}(x_3)$, which is discussed at the end of this subsection.

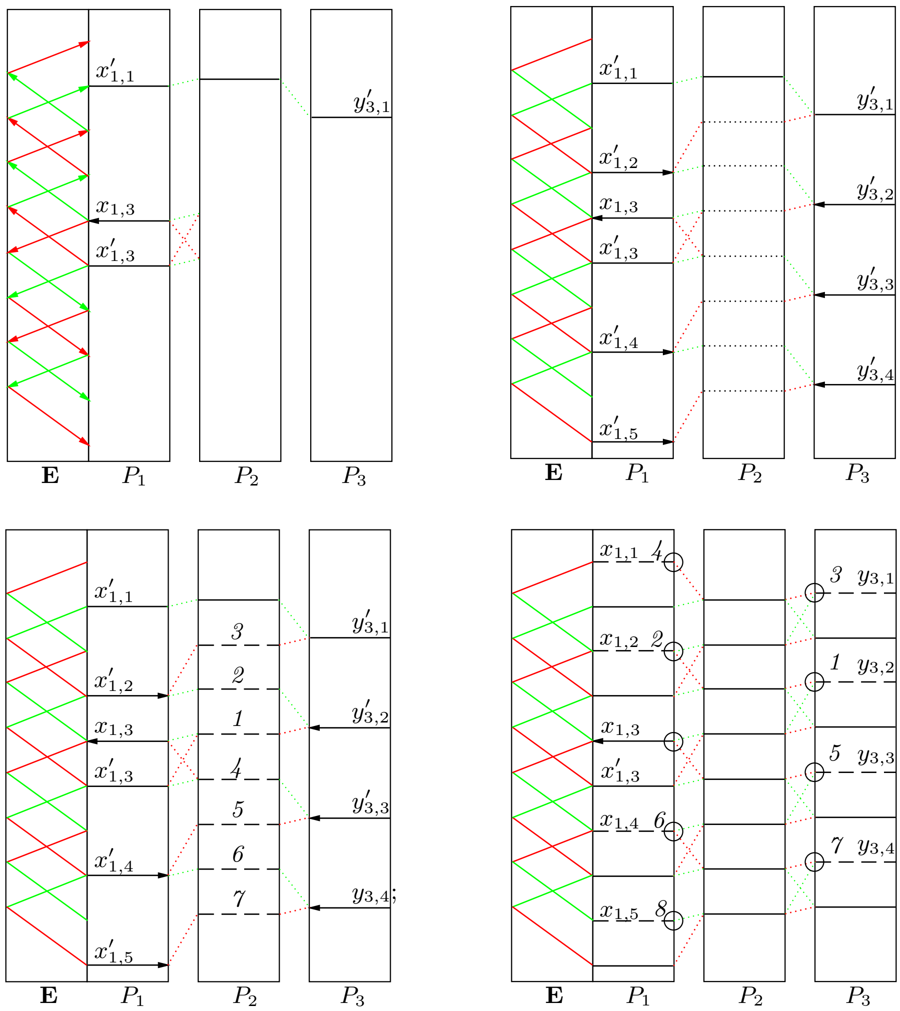


Fig. 3. The flow of PROCESS11SHOOT. Points x_1 and y_3 joined by a red (green, resp.) solid line satisfies $\mathbf{E}.\mathrm{E}(K_1, x_1) = y_3$ ($\mathbf{E}.\mathrm{E}(K_2, x_1) = y_3$, resp.), while points x_{i+1} and y_i joined by a red (green, resp.) dotted line satisfies $x_{i+1} = y_i \oplus k_1$ ($x_{i+1} = y_i \oplus k_2$, resp.). (top left) *Make-E-Chain-Phase*: S makes $2 \cdot 4t$ E-queries to form two E-chains. Depicted is a simple example with $4t = 4$, D querying $\mathrm{P1}^{-1}(y_1) \to x_1$ and S detecting $(1, x_1', y_1') : y_1' = y_1 \oplus k_1 \oplus k_2$ and taking (x_1, x_1') as $(y_{1,3}, y_{1,3}')$. The arrows of the solid directed lines indicate the directions of S's evaluation. However, note that some of the queries involved in this evaluation may already existed in *EQueries* and *Queries* before D querying $\mathrm{P1}^{-1}(y_1)$ – they may even formed completed path, e.g. the path for $(K_1, x_{1,1}, y_{3,1})$ in the figure. (top right) *Shoot-Growing-Phase*: S makes several 1- and 3-queries "grow out of the ground". Each such query will form a shoot, but the other query of this shoot remains missing. In this phase, pre-existing 1- and 3-queries stay invariant, while the newly created 1-queries (3-queries, resp) have $dir = \rightarrow$ ($dir = \leftarrow$, resp.). The points joined by black dotted lines already satisfy the relation $\mathrm{P2}(x_2) = y_2$ (logically), and this will be made explicit in the next phase. (bottom left) *Fill-in-Rung-Phase*: S fills proper AD-2-queries between the peaks of the "incomplete shoots". The numbers on these AD-2-queries indicate the order for S creating them. (bottom right) *Shoot-Completing-Phase*: S completes the "incomplete shoots" with AD-1- and AD-3-queries (dashed lines). The numbers on these AD-queries indicate the order for S creating them.

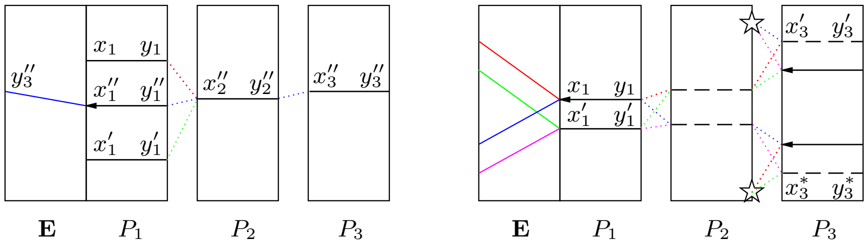


Fig. 4. Related-key boomerang structures under more than two keys. Points x_1 and y_3 joined by red, green, blue, and magenta solid lines satisfy the relations $\mathrm{E}(K_1, x_1) = y_3$, $\mathrm{E}(K_2, x_1) = y_3$, $\mathrm{E}(K_3, x_1) = y_3$, and $\mathrm{E}(K_4, x_1) = y_4$ respectively. Points y_i and x_{i+1} joined by red, green, blue, and magenta dotted line satisfy the relations $x_{i+1} = y_i \oplus k_1$, $x_{i+1} = y_i \oplus k_2$, $x_{i+1} = y_i \oplus k_3$, and $x_{i+1} = y_i \oplus k_4$ resp. Black (arrowed) lines are (directed) P-queries (may be internally created ones), while black dashed lines indicate AD-queries. (Left) Two such structures share a 1-query/a complete path; (Right) The bad situation due to $k_1 \oplus k_2 \oplus k_3 \oplus k_4 = 0$. Initially, S detects two 11-shoots $(1, x_1, \{K_1, K_2\})$ and $(1, x_1, \{K_3, K_4\})$. Later after S adapts and completes a 33-shoot $(3, y_3', \{K_1, K_2\})$, it detects an additional one $(3, y_3', \{K_3, K_4\})$; after S completes $(3, y_3^*, \{K_3, K_4\})$, it detects $(3, y_3^*, \{K_1, K_2\})$, cf. the two pentagrams.

Upon a New Query $\mathrm{P1}(x_1)$: this query may set off more than one 31-TPs, and S should take care of each. Thus S first gets $y_1 \leftarrow \mathbf{R}.\mathrm{P1}(x_1)$ (and creates $(1, x_1, y_1, \rightarrow)$), and then for each pair $((K, k), (3, x_3, y_3)) \in \mathit{HQueries} \times \mathit{Queries}$ it checks whether $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$ (via a procedure CHECK), and creates an AD-2-query $(2, y_1 \oplus k, x_3 \oplus k)$ if the checking operation returns positively. After all these, if S does not abort, then it returns $P_1(x_1)$ to answer D.

Upon a new query $\mathrm{P3}^{-1}(y_3)$, S behaves similarly by symmetry.

Upon a New Query $\mathrm{P2}(x_2)$: if $x_2 \oplus k \notin P_1^{-1}$ holds for all $k \in \mathcal{Z}$, then this query sets off no 12-TP, and S simply gets $y_2 \leftarrow \mathbf{P}.\mathrm{P2}(x_2)$ and creates $(2, x_2, y_2, \rightarrow)$. Otherwise, S detects new 12-TPs. However, to unify S's behaviors, we let S use the mechanism for 13-TPs to handle these 12-TPs. More clearly, due to the rhizome-mechanism, w.h.p. there exists at most one $(K, k) \in \mathit{HQueries}$ such that $x_2 \oplus k \in P_1^{-1}$. Thus we let S first compute $x_1 \leftarrow P_1^{-1}(k \oplus x_2)$, second query $y_3 \leftarrow \mathbf{E}.\mathrm{E}(K, x_2)$, and then functions as if D just queries $\mathrm{P3}^{-1}(y_3)$. (In the pseudocode, S make a call to its "inner $\mathrm{P3}^{-1}$" interface $\mathrm{P3In}^{-1}(y_3)$ to preform this.) Clearly, if S does not abort, then an AD-2-query $(2, x_2, y_2, \perp)$ will be created. In each case, S finally returns $P_2(x_2)$ if non-aborting.

The behaviors upon new $\mathrm{P2}^{-1}(y_2)$ are similar by symmetry.

Upon a New Query $\mathrm{H}(K)$: this query may lead S to detecting some H-TPs. Thus S first gets $k \leftarrow \mathbf{R}.\mathrm{H}(K)$ and creates (K, k), and then for each pair $((1, x_1, y_1), (3, x_3, y_3)) \in \mathit{Queries}$ such that $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$ (checked via CHECK), it creates an AD-2-query $(2, y_1 \oplus k, x_3 \oplus k)$.

Upon a New Query $\mathrm{P1}^{-1}(y_1)$: S collects the newly formed shoots and 21-TPs, and push them into two queues, and then starts a recursive chain-completion process. More clearly, S first gets $x_1 \leftarrow \mathbf{R}.\mathrm{P1}^{-1}(y_1)$ and creates $(1, x_1, y_1, \leftarrow)$, and then calls a procedure $\mathrm{CollectTP}(1, x_1, y_1)$, which (roughly speaking) functions as follows:

- for each distinct pair $(K, k), (K', k') \in \mathit{HQueries}$, if $y_1' = y_1 \oplus k \oplus k' \in P_1^{-1}$, then S detects a new 11-shoot, and pushes a 3-tuple $(1, x_1, \{K, K'\})$ into the first queue *ShootQueue*. We stress that the third coordinate of this tuple is a *set*, i.e. the order of K and K' does not matter;
- for each $(K, k) \in \mathit{HQueries}$ that is not involved in any 11-shoots – formally, $\forall (K', k') \neq (K, k) \in \mathit{HQueries}$: $y_1 \oplus k \oplus k' \notin P_1^{-1}$, – if $y_1 \oplus k \in P_2$, then S detects a new 21-TP, and pushes a 3-tuple $(1, x_1, K)$ into the second queue *MidTPQueue*.

After CollectTP returns, S calls EmptyQueue, which pops tuples from the two queues and passes control to the other procedures to tackle them. For each tuple $(1, x_1, \{K_1, K_2\})$ ($(3, y_3, \{K_1, K_2\})$, resp.) popped from *ShootQueue*, it makes a call to $\mathrm{Process11Shoot}(x_1, P_1(x_1), K_1, K_2)$ ($\mathrm{Process33Shoot}(P_3^{-1}(y_3), y_3, K_1, K_2)$, resp.) if $(1, x_1, \{K_1, K_2\}) \notin \mathit{ProcessedShoot}$ ($(3, y_3, \{K_1, K_2\}) \notin \mathit{ProcessedShoot}$, resp.); for each tuple $(1, x_1, K)$ ($(3, y_3, K)$, resp.) popped from *MidTPQueue*, it makes a call to $\mathrm{Process21TP}(x_1, P_1(x_1), K)$ ($\mathrm{Process23TP}$ $(P_3^{-1}(y_3), y_3, K)$, resp.) if $(1, K, x_1) \notin \mathit{Completed}$ ($(3, K, P_3^{-1}(y_3)) \notin \mathit{Completed}$, resp.).

Note that the arguments of PROCESS21TP identify a partial-chain $x_1 - y_1 - x_2 - y_2 - x_3$, where $y_1 = P_1(x_1)$, $x_2 = y_1 \oplus k$, $y_2 = P_2(x_2)$, and $x_3 = y_2 \oplus k$. PROCESS21TP$(x_1, P_1(x_1), K)$ completes this chain by first querying $y_3 \leftarrow \mathbf{E}.\mathrm{E}(K, x_1)$ and then creating an AD-3-query $(3, x_3, y_3, \perp)$. It finally calls COLLECTTP$(3, x_3, y_3)$ to collect the TPs newly set off by this AD-3-query, and push them into *ShootQueue* and *MidTPQueue* respectively:

– for each distinct pair $(K, k), (K', k') \in HQueries$, if $x'_3 = x_3 \oplus k \oplus k' \in P_3$, then S detects a 33-shoot and pushes a 3-tuple $(3, y_3, \{K, K'\})$ into *ShootQueue*;
– for each $(K, k) \in HQueries$ that is not involved in any 33-shoots, if $x_3 \oplus k \in P_2^{-1}$, then S detects a new 23-TP, and pushes a 3-tuple $(3, y_3, K)$ into *MidTPQueue*.

The flow of PROCESS23TP is similar by symmetry, leading to creating an AD-1-query. Similarly to the shoots, we will use G_2 *processing a 21-/23-TP (or MidTP)* to (indifferently) refer to G_2 executing PROCESS21TP/PROCESS23TP.

On the other hand, the PROCESSSHOOT-procedures have been introduced in subsection 4.4. However, we stress that for each AD-1- and AD-3-query newly created by these calls, a call to COLLECTTP would be made to collect (and enqueue) the newly detected TPs. Thus EMPTYQUEUE keeps popping tuples from the two queues and calling the four procedures, till both of them are empty.

The behaviors upon new P3(x_3) are similar by symmetry, to wit, S first creates $(3, x_3, y_3, \rightarrow)$ with $y_3 = \mathbf{R}.\mathrm{P3}(x_3)$, and then calls COLLECTTP$(3, x_3, y_3)$, and then performs as described.

In the rest part, such a tuple $(1, x_1, \{K, K'\})$ is also called a 11-shoot, and $(3, y_3, \{K, K'\})$ is a 33-shoot. For the former, x_1 is its *root* and $P_1(x_1)$ is its *peak*, while for the latter, y_3 and $P_3^{-1}(y_3)$ are its root and peak respectively. Moreover, a tuple $(1, x_1, K)$ is called a *21-TP* with x_1 being the root and $P_1(x_1)$ being the peak, while a tuple $(3, y_3, K)$ is a *23-TP* with root y_3 and peak $P_3^{-1}(y_3)$. Finally, calls to the four procedures PROCESS21TP, PROCESS23TP, PROCESS11SHOOT, and PROCESS33SHOOT are *chain-reaction calls.*

SHOOTS HAVE HIGHER PRIORITY THAN MIDTPS One may note that in our chain-detection mechanism, shoots have higher priority than MidTPs. For example, in COLLECTTP$(1, x_1, y_1)$ as described, once there exist two distinct $(K, k), (K', k') \in HQueries : y'_1 = y_1 \oplus k \oplus k' \in P_1^{-1}$, then S only pushes a 11-shoot $(1, x_1, \{K, K'\})$ into *ShootQueue*, and ignores the possibly existing 21-TPs $(1, x_1, K)$ and $(1, x_1, K')$. The considerations are as follows:

(i) First, since D's queries have formed such a shoot, S has to prepare the "rhizome structure" (cf. Fig. 3), as otherwise S would not be ready to deal with D's future queries in this structure and risk of being "trapped";
(ii) Second, it can be seen that once the structure around the shoot $(1, x_1, \{K, K'\})$ is prepared as expected, then the two possibly 21-TPs $(1, x_1, K)$ and $(1, x_1, K')$ would have been in completed paths, and thus do not need to be separately considered.

Denote by G_1 the simulated system formed by $S^{\mathbf{R}}$ and $\mathbf{E}$, and by G_3 the real system formed by EMR_3^* and $\mathbf{R}$. To simplify notations and highlight randomness sources, we will use $G_1(\mathbf{E}, \mathbf{R})$ and $G_3(\mathbf{R})$ to refer to the systems respectively.

5.2 Pseudocode of the Simulator

Following [ABD+13a], we also denote by $m\mathcal{Z}$ the m-fold direct $\oplus$-sum $\mathcal{Z} \oplus \ldots \oplus \mathcal{Z}$ of $\mathcal{Z}$. Then the following pseudocode implements the simulated system G_1 along with the intermediate system G_2 (cf. Section 6). When a line has a boxed variant next to it, G_1 uses the original code, whereas G_2 uses the boxed one. Additionally, the underlined red sentences only exist in G_2. Indeed, the code for G_1 is exactly the code for S.

Simulated System $G_1(\mathbf{E}, \mathbf{R})$ $\boxed{\textbf{Intermediate System } G_2(\mathbf{E}, \mathbf{R})}$

Variables

Sets *Queries* and *HQueries*, initialized to $\emptyset$ // Sets for history.
Set *EQueries*, initialized to $\emptyset$ // bookkeeping set for G_2
Queues *ShootQueue*, *MidTPQueue* // Queue for (detected) bamboo shoots and middle TPs, resp.
Sets *Completed*, *ProcessedShoots*, initialized to $\emptyset$ // Set of completed chains and processed shoots, resp.
Integer *qnum*, initialized to 1.
Set *AD2Edges*, initialized to $\emptyset$ // Set of 2-edges formed by AD-2-queries.
Set *DUShoots*, initialized to $\emptyset$ // Set of D-unaware shoots.
Set *Border*, initialized to $\emptyset$ // Set of x_1 and y_3 values that lie at the endpoints of rhizomes.

Integer $\mathit{cycleStartNum}$

// The following four enc/decryption procedures only exist in G_2. In G_1, the interfaces E and E^{-1} are simply provided by **E**.

public procedure $\mathrm{E}(K, x_1)$ // G_2
CheckDUnaware$(x_1, X1)$
$y_3 \leftarrow$ EIn(K, x_1)
RemoveDUShoots$(3, y_3)$
return y_3

private procedure EIn(K, x_1) // G_2
if $x_1 \notin \mathit{ETable}[K]$ **then**
$y_3 \leftarrow \mathbf{E}.\mathrm{E}(K, x_1)$
if $y_3 \in P_3^{-1}$ **then abort**
if $\exists K' \neq K : y_3 \in \mathit{ETable}[K']^{-1}$ **then abort**
$\mathit{EQueries} \leftarrow \mathit{EQueries} \cup \{(K, x_1, y_3, \rightarrow, \mathit{qnum})\}$
$\mathit{qnum} \leftarrow \mathit{qnum} + 1$
return $\mathit{ETable}[K](x_1)$

private procedure RemoveDUShoots(i, z) // G_2
if $i = 1$ **then**
Let $st = (1, \{(z, y_1), (z', y_1')\}) \in \mathit{DUShoots}$.
$\mathit{DUShoots} \leftarrow \mathit{DUShoots} \setminus \{st\}$
else // $i = 3$
Let $st = (3, \{(x_3, z), (x_3', z')\}) \in \mathit{DUShoots}$.
$\mathit{DUShoots} \leftarrow \mathit{DUShoots} \setminus \{st\}$

public procedure $\mathrm{E}^{-1}(K, y_3)$ // G_2
CheckDUnaware$(y_3, Y3)$
$x_1 \leftarrow \mathrm{EIn}^{-1}(K, y_3)$
RemoveDUShoots$(1, x_1)$
return x_1

private procedure $\mathrm{EIn}^{-1}(K, y_3)$ // G_2
if $y_3 \notin \mathit{ETable}[K]^{-1}$ **then**
$x_1 \leftarrow \mathbf{E}.\mathrm{E}^{-1}(K, y_3)$
if $x_1 \in P_1$ **then abort**
if $\exists K' \neq K : x_1 \in \mathit{ETable}[K']$ **then abort**
$\mathit{EQueries} \leftarrow \mathit{EQueries} \cup \{(K, x_1, y_3, \leftarrow, \mathit{qnum})\}$
$\mathit{qnum} \leftarrow \mathit{qnum} + 1$
return $\mathit{ETable}[K]^{-1}(y_3)$

private procedure Assert(fact) // G_2
if $\neg \mathit{fact}$ **then**
abort

private procedure Check(K, x_1, y_3)
return $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$ $\boxed{\textbf{return } \mathit{ETable}[K](x_1) = y_3}$

private procedure CheckDUnaware(z, tag) // G_2
if DAwareness$(z, \mathit{tag}) = 0$ **then abort**

private procedure DAwareness(z, tag) // G_2
if $\mathit{tag} = X1$ **and** $\exists (1, \{(z, y_1), (x_1', y_1')\}) \in \mathit{DUShoots}$
then return 0
if $\mathit{tag} = Y1$ **then**
if $\exists (1, \{(x_1, y_1), (x_1', y_1')\}) \in \mathit{DUShoots}$:
$y_1 \oplus z \in 2\mathcal{Z}$ **or** $y_1' \oplus z \in 2\mathcal{Z}$ **then**
return 0
if $\mathit{tag} = X2$ **then**
if $\exists (1, \{(x_1, y_1), (x_1', y_1')\}) \in \mathit{DUShoots}$:
$y_1 \oplus z \in \mathcal{Z}$ **or** $y_1' \oplus z \in \mathcal{Z}$ **then**
return 0
if $\mathit{tag} = Y2$ **then**
if $\exists (3, \{(x_3, y_3), (x_3', y_3')\}) \in \mathit{DUShoots}$:
$x_3 \oplus z \in \mathcal{Z}$ **or** $x_3' \oplus z \in \mathcal{Z}$ **then**
return 0
if $\mathit{tag} = X3$ **then**
if $\exists (3, \{(x_3, y_3), (x_3', y_3')\}) \in \mathit{DUShoots}$:
$x_3 \oplus z \in 2\mathcal{Z}$ **or** $x_3' \oplus z \in 2\mathcal{Z}$ **then**
return 0
if $\mathit{tag} = Y3$ **and** $\exists (3, \{(x_3, z), (x_3', y_3')\}) \in \mathit{DUShoots}$
then return 0
return 1

public procedure $\mathrm{H}(K)$
if $K \in \mathit{HTable}$ **then return** $\mathit{HTable}(K)$
$k \leftarrow \mathbf{R}.\mathrm{H}(K)$
if $k \in \mathcal{Z}$ **then abort**
if there exist three distinct $k', k'', k''' \in \mathcal{Z} : k \oplus k' \oplus k'' \oplus k''' = 0$ **then abort**
if $\exists i, y_i \in P_i^{-1}, x_{i+1} \in P_{i+1} : y_i \oplus x_{i+1} \in (k \oplus 4\mathcal{Z}) \cup \{k\}$
or $\exists i, x_i, x_i' \in P_i : x_i \oplus x_i' \in k \oplus 5\mathcal{Z}$
or $\exists i, y_i, y_i' \in P_i^{-1} : y_i \oplus y_i' \in k \oplus 5\mathcal{Z}$ **then abort**
$\mathit{HQueries} \leftarrow \mathit{HQueries} \cup \{(K, k, \mathit{qnum})\}$
$\mathit{qnum} \leftarrow \mathit{qnum} + 1$
// Deal with H-TPs:
foreach $(1, x_1, y_1), (3, x_3, y_3) \in \mathit{Queries} \times \mathit{Queries}$ **do**
if Check$(K, x_1, y_3) =$ **true then**
Take the E-query $(K, x_1, y_3, \mathit{edir}, \mathit{enum})$ from $\mathit{EQueries}$
Adapt$(2, y_1 \oplus k, x_3 \oplus k, \mathit{edir}, \mathit{enum})$
// In G_1, S uses arbitrary "dummy" edir and enum for this call. Same for the other calls to Adapt$(2, \ldots)$.
Assert$(\forall k' \in \mathcal{Z} \backslash \{k\} : x_2 \oplus k' \notin P_1^{-1}$ and $y_2 \oplus k' \notin P_3)$ // The newly created 2-query would not trigger 12- and 32-TP.
UpdateCompleted$(1, K, x_1)$

// Update the set of AD-2-edges:
foreach AD-2-query $(2, x_2, y_2, \perp) \in \mathit{Queries}$ **do**
  Arbitrarily chooses $(K', k') \in \mathit{HQueries}, K' \neq K$.
  Assert(the edge $(x_2 \oplus k', y_2 \oplus k')$ is in $\mathit{AD2Edges}$)
  Takes $(x_2 \oplus k', y_2 \oplus k', \mathit{ad2dir}, \mathit{ad2num}) \in \mathit{AD2Edges}$.
  $\mathit{AD2Edges} \leftarrow \mathit{AD2Edges} \cup \{(x_2 \oplus k, y_2 \oplus k, k, \mathit{ad2dir}, \mathit{ad2num})\}$
return $\mathit{HTable}(K)$

private procedure RandAssign(i, z, δ) // The term "random assign" is from [LS13].
if $\delta = +$ **then**
  $z' \leftarrow \mathbf{R}.\mathrm{P}i(z)$
  if $z' \in P_i^{-1}$ **then abort**
  AddQuery$(i, z, z', \rightarrow, \emptyset)$
  return z'
else // $\delta = -$
  $z' \leftarrow \mathbf{R}.\mathrm{P}i^{-1}(z)$
  if $z' \in P_i$ **then abort**
  AddQuery$(i, z, z', \leftarrow, \emptyset)$
  return z'

// Create the record of a query.
private procedure AddQuery$(i, x_i, y_i, \mathit{dir})$
if $(i, \mathit{dir}) \in \{(1, \rightarrow), (2, \rightarrow)\} \wedge \exists (i+1, x_{i+1}, y_{i+1}) \in \mathit{Queries} : y_i \oplus x_{i+1} \in 5\mathcal{Z}$ **then**
  abort // Early-abortions in G_2. Same for the below.
if $(i, \mathit{dir}) = (3, \rightarrow) \wedge \exists K : y_3 \in \mathit{ETable}[K]^{-1}$ **then abort**
if $\mathit{dir} = \rightarrow \wedge \exists (i, x_i', y_i') \in \mathit{Queries} : y_i \oplus y_i' \in 6\mathcal{Z}$ **then abort**
if $(i, \mathit{dir}) \in \{(2, \leftarrow), (3, \leftarrow)\} \wedge \exists (i-1, x_{i-1}, y_{i-1}) \in \mathit{Queries} : y_{i-1} \oplus x_i \in 5\mathcal{Z}$ **then abort**
if $(i, \mathit{dir}) = (1, \leftarrow) \wedge \exists K : x_1 \in \mathit{ETable}[K]$ **then abort**
if $\mathit{dir} = \leftarrow \wedge \exists (i, x_i', y_i') \in \mathit{Queries} : x_i \oplus x_i' \in 6\mathcal{Z}$ **then abort**
$\mathit{Queries} \leftarrow \mathit{Queries} \cup \{(i, x_i, y_i, \mathit{dir}, \mathit{qnum})\}$
$\mathit{qnum} \leftarrow \mathit{qnum} + 1$
if $|P_1| > 13(q_e + q_p) \cdot q_p^2$ **or** $|P_3| > 13(q_e + q_p) \cdot q_p^2$ **then abort**

public procedure P1$^{-1}(y_1)$
  CheckDUnaware$(y_1, Y1)$
  if $y_1 \in P_1^{-1}$ **then return** $P_1^{-1}(y_1)$
  $\mathit{cycleStartNum} \leftarrow \mathit{qnum}$
  $x_1 \leftarrow$ RandAssign$(1, y_1, -)$
  CollectTP$(1, x_1, y_1)$
  EmptyQueue()

public procedure P3(x_3)
  CheckDUnaware$(x_3, X3)$
  if $x_3 \in P_3$ **then return** $P_3(x_3)$
  $\mathit{cycleStartNum} \leftarrow \mathit{qnum}$
  $y_3 \leftarrow$ RandAssign$(3, x_3, +)$
  CollectTP$(3, x_3, y_3)$
  EmptyQueue()

private procedure CollectTP(i, z, z')
if $i = 1$ **then**
  Take (z, z') as (x_1, y_1), a new 1-query.
  foreach two distinct $(K, k)(K', k') \in \mathit{HQueries}$ **do**
    if $(1, x_1, \{K, K'\}) \in \mathit{ProcessedShoot}$ **then**
      continue
    $y_1' \leftarrow y_1 \oplus k \oplus k'$
    if $y_1' \notin P_1^{-1}$ **then continue**
    Take the 1-query $(1, x_1', y_1', \mathit{num}')$ from $\mathit{Queries}$
    Assert$(x_1' \notin \mathit{Border})$
    Assert$(\mathit{num}' < \mathit{cycleStartNum})$
    $\mathit{ShootQueue}$.Enqueue$(1, x_1, \{K, K'\})$
  foreach $(K, k) \in \mathit{HQueries}$ **do**
    if $\exists k' \in \mathcal{Z} \setminus \{k\} : y_1 \oplus k \oplus k' \in P_1^{-1}$ **then**
      continue
    $x_2 \leftarrow y_1 \oplus k$
    if $x_2 \notin P_2$ **then continue**
    Obtain the 2-query $(2, x_2, y_2, \mathit{num}_2)$ from $\mathit{Queries}$
    Assert$(\mathit{num}_2 < \mathit{cycleStartNum})$
    $\mathit{MidTPQueue}$.Enqueue$(1, x_1, K)$
else // $i = 3$
  Take (z, z') as (x_3, y_3), a new 3-query.
  foreach two distinct $(K, k)(K', k') \in \mathit{HQueries}$ **do**
    if $(3, y_3, \{K, K'\}) \in \mathit{ProcessedShoot}$ **then**
      continue
    $x_3' \leftarrow x_3 \oplus k \oplus k'$
    if $x_3' \notin P_3$ **then continue**
    Obtain the 3-query $(3, x_3', y_3', \mathit{num}')$ from $\mathit{Queries}$
    Assert$(y_3' \notin \mathit{Border})$
    Assert$(\mathit{num}' < \mathit{cycleStartNum})$
    $\mathit{ShootQueue}$.Enqueue$(3, y_3, \{K, K'\})$
  foreach $(K, k) \in \mathit{HQueries}$ **do**
    if $\exists k' \in \mathcal{Z} \setminus \{k\} : x_3 \oplus k \oplus k' \in P_3$ **then**
      continue
    $y_2 \leftarrow x_3 \oplus k$
    if $y_2 \notin P_2^{-1}$ **then continue**
    Obtain the 2-query $(2, x_2, y_2, \mathit{num}_2)$ from $\mathit{Queries}$
    Assert$(\mathit{num}_2 < \mathit{cycleStartNum})$
    $\mathit{MidTPQueue}$.Enqueue$(3, y_3, K)$

private procedure EmptyQueue()

do
  while $\neg \mathit{ShootQueue}.\textsc{Empty}()$ **do**
    $(i, rt, \{K_1, K_2\}) \leftarrow \mathit{ShootQueue}.\textsc{Dequeue}()$
    if $(i, rt, \{K_1, K_2\}) \in \mathit{ProcessedShoots}$ **then**
      continue
    // Depending on the type of this shoot:
    if $i = 1$ **then**
      $\textsc{Process11Shoot}(rt, P_1(rt), K_1, K_2)$
    else // $i = 3$
      $\textsc{Process33Shoot}(P_3^{-1}(rt), rt, K_1, K_2)$
  while $\neg \mathit{MidTPQueue}.\textsc{Empty}()$ **do**
    $(i, rt, K) \leftarrow \mathit{MidTPQueue}.\textsc{Dequeue}()$
    if $i = 1$ **then**
      if $(1, K, \mathit{root}) \in \mathit{Completed}$ **then continue**
      $\textsc{Process21TP}(rt, P_1(rt), K)$
    else // $i = 3$
      if $(3, K, P_3^{-1}(rt)) \in \mathit{Completed}$ **then continue**
      $\textsc{Process23TP}(P_3^{-1}(rt), rt, K)$
while$(\neg \mathit{ShootQueue}.\textsc{Empty}())$

private procedure $\textsc{Process11Shoot}(x_1, y_1, K_1, K_2)$
  $k_1 \leftarrow \mathit{HTable}(K_1)$, $k_2 \leftarrow \mathit{HTable}(K_2)$, $\mathit{NewDUShootSet} \leftarrow \emptyset$
  Take (x_1, y_1) as $(x_{1,t+1}, y_{1,t+1})$
  $y'_{1,t+1} \leftarrow y_{1,t+1} \oplus k_1 \oplus k_2$, $x'_{1,t+1} \leftarrow P_1^{-1}(y'_{1,t+1})$
  // When $q_e + q_p$ is odd then let $q_e + q_p = 2t - 3$; else let $q_e + q_p = 2t - 4$.
  // *Make-E-Chain-Phase*: make $4t$ pairs of queries to $\mathbf{E}$ (in G_1), or $\textsc{EIn}$ and $\textsc{EIn}^{-1}$ (in G_2).
  // $x'_{1,1} \xleftarrow{E_{K_2}^{-1}} \ldots \xleftarrow{E_{K_1}} x'_{1,t} \xleftarrow{E_{K_2}^{-1}} y'_{3,t} \xleftarrow{E_{K_1}} x'_{1,t+1} \xrightarrow{E_{K_2}} y'_{3,t+1} \xrightarrow{E_{K_1}^{-1}} x'_{1,t+2} \xrightarrow{E_{K_2}} \ldots \xrightarrow{E_{K_1}^{-1}} x'_{1,2t+1}$
  // $x_{1,1} \xleftarrow{E_{K_1}^{-1}} \ldots \xleftarrow{E_{K_2}} x_{1,t} \xleftarrow{E_{K_1}^{-1}} y_{3,t} \xleftarrow{E_{K_2}} x_{1,t+1} \xrightarrow{E_{K_1}} y_{3,t+1} \xrightarrow{E_{K_2}^{-1}} x_{1,t+2} \xrightarrow{E_{K_1}} \ldots \xrightarrow{E_{K_2}^{-1}} x_{1,2t+1}$
  for i **from** t **to** 1 **do**
    if $x'_{1,i+1} \notin \mathit{ETable}[K_1]$ **then**
      $\mathit{NewDUShootSet} \leftarrow \mathit{NewDUShootSet} \cup \{(3, i)\}$
    $y'_{3,i} \leftarrow \mathbf{E}.\mathrm{E}(K_1, x'_{1,i+1})$ $\boxed{y'_{3,i} \leftarrow \textsc{EIn}(K_1, x'_{1,i+1})}$
    if $y'_{3,i} \notin \mathit{ETable}[K_2]^{-1}$ **then**
      $\mathit{NewDUShootSet} \leftarrow \mathit{NewDUShootSet} \cup \{(1, i)\}$
    $x'_{1,i} \leftarrow \mathbf{E}.\mathrm{E}^{-1}(K_2, y'_{3,i})$ $\boxed{x'_{1,i} \leftarrow \textsc{EIn}^{-1}(K_2, y'_{3,i})}$
  for i **from** $t+1$ **to** $2t$ **do**
    if $x'_{1,i} \notin \mathit{ETable}[K_2]$ **then**
      $\mathit{NewDUShootSet} \leftarrow \mathit{NewDUShootSet} \cup \{(3, i)\}$
    $y'_{3,i} \leftarrow \mathbf{E}.\mathrm{E}(K_2, x'_{1,i})$ $\boxed{y'_{3,i} \leftarrow \textsc{EIn}(K_2, x'_{1,i})}$
    if $y'_{3,i} \notin \mathit{ETable}[K_1]^{-1}$ **then**
      $\mathit{NewDUShootSet} \leftarrow \mathit{NewDUShootSet} \cup \{(1, i+1)\}$
    $x'_{1,i+1} \leftarrow \mathbf{E}.\mathrm{E}^{-1}(K_1, y'_{1,i})$ $\boxed{x'_{1,i+1} \leftarrow \textsc{EIn}^{-1}(K_1, y'_{1,i})}$
  for i **from** t **to** 1 **do**
    $y_{3,i} \leftarrow \mathbf{E}.\mathrm{E}(K_1, x_{1,i+1})$ $\boxed{y_{3,i} \leftarrow \textsc{EIn}(K_1, x_{1,i+1})}$
    $x_{1,i} \leftarrow \mathbf{E}.\mathrm{E}^{-1}(K_2, y_{3,i})$ $\boxed{x_{1,i} \leftarrow \textsc{EIn}^{-1}(K_2, y_{3,i})}$
  for i **from** $t+1$ **to** $2t$ **do**
    $y_{3,i} \leftarrow \mathbf{E}.\mathrm{E}(K_2, x_{1,i})$ $\boxed{y_{3,i} \leftarrow \textsc{EIn}(K_2, x_{1,i})}$
    $x_{1,i+1} \leftarrow \mathbf{E}.\mathrm{E}^{-1}(K_1, y_{1,i})$ $\boxed{x_{1,i+1} \leftarrow \textsc{EIn}^{-1}(K_1, y_{1,i})}$
  // *Shoot-Growing-Phase*: make the shoots "grow out of the ground".
  foreach $x'_{1,i}$ **do**
    if $x'_{1,i} \notin P_1$ **then**
      $y'_{1,i} \leftarrow \textsc{RandAssign}(1, x'_{1,i}, +)$
      foreach $(K, k, \overline{x_3}, \overline{y_3}) : (K, k) \in \mathit{HQueries}$ **and** $k \neq k_1, k_2$ **and** $(3, \overline{x_3}, \overline{y_3}) \in \mathit{Queries}$ **do**
        if $\textsc{Check}(K, x'_{1,i}, \overline{y_3}) = \textbf{true}$ **then**
          $\overline{x_2} \leftarrow y'_{1,i} \oplus k$, $\overline{y_2} \leftarrow \overline{x_3} \oplus k$
          Take $(K, x'_{1,i}, \overline{y_3}, \mathit{edir}, \mathit{enum})$ from $\mathit{EQueries}$
          $\textsc{Adapt}(2, \overline{x_2}, \overline{y_2}, \mathit{edir}, \mathit{enum})$ // "Dummy" edir and enum in G_1.
          $\textsc{UpdateCompleted}(1, K, x'_{1,i})$
          $\textsc{Assert}(\nexists k' \neq k : \overline{x_2} \oplus k' \in P_1^{-1})$ // The new 2-query would not trigger 12-TP.
          foreach $k' \neq k : \overline{y_2} \oplus k' \in P_3$ **do** // No additional 32-TP has to be considered.
            $\overline{y_3}' \leftarrow P_3(\overline{y_2} \oplus k')$
            $\textsc{Assert}(\overline{y_3}' \notin \mathit{Border} \wedge \exists (3, \overline{y_3}', \{K, K'\}) \in \mathit{ShootQueue})$
  foreach $y'_{3,i}$ **do**

```
    if y'_{3,i} ∉ P_3^{-1} then
      x'_{3,i} ← RandAssign(3, y'_{3,i}, −)
      foreach (K, k, x̄_1, ȳ_1) : (K, k) ∈ HTable and k ≠ k_1, k_2 and (1, x̄_1, ȳ_1) ∈ Queries do
        if Check(K, x̄_1, y'_{3,i}) = true then
          x̄_2 ← x̄_1 ⊕ k, ȳ_2 ← x'_{3,i} ⊕ k
          Take (K, x̄_1, y'_{3,i}, edir, enum) from EQueries
          Adapt(2, x̄_2, ȳ_2, edir, enum) // "Dummy" edir and enum in G_1.
          UpdateCompleted(1, K, x̄_1)
          Assert(∄k' ≠ k : ȳ_2 ⊕ k' ∈ P_3)
          foreach k' ≠ k : x̄_2 ⊕ k' ∈ P_1^{-1} do
            x̄_1' ← P_1^{-1}(x̄_2 ⊕ k')
            Assert(x̄_1' ∉ Border ∧ ∃(1, x̄_1', {K, K'}) ∈ ShootQueue)
  // Fill-in-Rung-Phase: fill in rungs with AD-2-queries
  for i from t to 1 do
    // Consider (x'_{3,i}, y'_{3,i})
    if x'_{3,i} ⊕ k_1 ∈ P_2^{-1} then
      Assert((3, K_1, x'_{3,i}) ∈ Completed) // If x'_{3,i} ⊕ k_1 has been occupied then the chain has been completed.
    else
      y'_{2,2i} ← x'_{3,i} ⊕ k_1, x'_{2,2i} ← y'_{1,i} ⊕ k_1
      Take (K_1, x'_{1,i+1}, y'_{3,i}, edir, enum) from EQueries
      Adapt(2, x'_{2,2i}, y'_{2,2i}, edir, enum) // "Dummy" edir and enum in G_1.
      UpdateCompleted(3, K_1, x'_{3,i})
      Assert(∄k ≠ k_1, k_2 : x'_{2,2i} ⊕ k ∈ P_1^{-1} or y'_{2,2i} ⊕ k ∈ P_3) // No new 12-/32-TP.
    if x'_{3,i} ⊕ k_2 ∈ P_2^{-1} then
      Assert((3, K_2, x'_{3,i}) ∈ Completed)
    else
      y'_{2,2i−1} ← x'_{3,i} ⊕ k_2, x'_{2,2i−1} ← y'_{1,i} ⊕ k_2
      Take (K_2, x'_{1,i}, y'_{3,i}, edir, enum) from EQueries
      Adapt(2, x'_{2,2i−1}, y'_{2,2i−1}, edir, enum) // "Dummy" edir and enum in G_1.
      UpdateCompleted(3, K_2, x'_{3,i})
      Assert(∄k ≠ k_1, k_2 : x'_{2,2i−1} ⊕ k ∈ P_1^{-1} or y'_{2,2i−1} ⊕ k ∈ P_3)
  for i from t + 1 to 2t do
    if x'_{3,i} ⊕ k_2 ∈ P_2^{-1} then
      Assert((3, K_2, x'_{3,i}) ∈ Completed)
    else
      y'_{2,2i−1} ← x'_{3,i} ⊕ k_2, x'_{2,2i−1} ← y'_{1,i} ⊕ k_2
      Take (K_2, x'_{1,i}, y'_{3,i}, edir, enum) from EQueries
      Adapt(2, x'_{2,2i−1}, y'_{2,2i−1}, edir, enum) // "Dummy" edir and enum in G_1.
      UpdateCompleted(3, K_2, x'_{3,i})
      Assert(∄k ≠ k_1, k_2 : x'_{2,2i−1} ⊕ k ∈ P_1^{-1} or y'_{2,2i−1} ⊕ k ∈ P_3)
    if x'_{3,i} ⊕ k_1 ∈ P_2^{-1} then
      Assert((3, K_1, x'_{3,i}) ∈ Completed)
    else
      y'_{2,2i} ← x'_{3,i} ⊕ k_1, x'_{2,2i} ← y'_{1,i} ⊕ k_1
      Take (K_1, x'_{1,i+1}, y'_{3,i}, edir, enum) from EQueries
      Adapt(2, x'_{2,2i}, y'_{2,2i}, edir, enum) // "Dummy" edir and enum in G_1.
      UpdateCompleted(3, K_1, x'_{3,i})
      Assert(∄k ≠ k_1, k_2 : x'_{2,2i} ⊕ k ∈ P_1^{-1} or y'_{2,2i} ⊕ k ∈ P_3)
  // Shoot-Completing-Phase: complete the bamboo shoots
  for i from t to 1 do
    // Consider first (3, x'_{3,i}, y'_{3,i}) and then (1, x'_{1,i}, y'_{1,i})
    x'_{3,i} ← P_3^{-1}(y'_{3,i}), x_{3,i} ← x'_{3,i} ⊕ k_1 ⊕ k_2
    if x_{3,i} ∈ P_3 or y_{3,i} ∈ P_3^{-1} then
      Assert(∃(3, y_{3,i}, {K, K'}) ∈ ProcessedShoots)
      Assert(P_3(x_{3,i}) = y_{3,i}) // If the values have been occupied then some relevant shoots have been processed.
    else
      if DAwareness(y'_{3,i}, Y3) = 0 then
        RemoveDUShoots(3, y'_{3,i})
      if (3, i) ∉ NewDUShootSet then
```

```
        CheckDUnaware(x_{3,i}, X3)
      Adapt(3, x_{3,i}, y_{3,i}, ⊥, ⊥)
      if ∃K ≠ K_1, K_2 : ETable[K]^{-1}(y_{3,i}) ∈ P_1 then
        Assert((3, K, x_{3,i}) ∈ Completed) // No new 31-TP is triggered.
      UpdateCompleted(3, K_2, x_{3,i})
      CollectTP(3, x_{3,i}, y_{3,i})
    ProcessedShoots ← ProcessedShoots ∪ {(3, y_{3,i}, {K_1, K_2}), (3, y'_{3,i}, {K_1, K_2})}
    y'_{1,i} ← P_1(x'_{1,i}), y_{1,i} ← y'_{1,i} ⊕ k_1 ⊕ k_2
    if x_{1,i} ∈ P_1 or y_{1,i} ∈ P_1^{-1} then
      Assert(∃(1, x_{1,i}, {K, K'}) ∈ ProcessedShoots)
      Assert(P_1(x_{1,i}) = y_{1,i})
    else
      if DAwareness(x'_{1,i}, X1) = 0 then
        RemoveDUShoots(1, x'_{1,i})
      if (1, i) ∉ NewDUShootSet then
        CheckDUnaware(y_{1,i}, Y1)
      Adapt(1, x_{1,i}, y_{1,i}, ⊥, ⊥)
      if ∃K ≠ K_1, K_2 : ETable[K](x_{1,i}) ∈ P_3^{-1} then
        Assert((1, K, x_{1,i}) ∈ Completed)
      UpdateCompleted(1, K_1, x_{1,i})
      CollectTP(1, x_{1,i}, y_{1,i})
    ProcessedShoots ← ProcessedShoots ∪ {(1, x_{1,i}, {K_1, K_2}), (1, x'_{1,i}, {K_1, K_2})}
  for i from t + 1 to 2t do
    // First (3, x'_{3,i}, y'_{3,i}) and then (1, x'_{1,i+1}, y'_{1,i+1})
    x'_{3,i} ← P_3^{-1}(y'_{3,i}), x_{3,i} ← x'_{3,i} ⊕ k_1 ⊕ k_2
    if x_{3,i} ∈ P_3 or y_{3,i} ∈ P_3^{-1} then
      Assert(∃(3, y_{3,i}, {K, K'}) ∈ ProcessedShoots)
      Assert(P_3(x_{3,i}) = y_{3,i})
    else
      if DAwareness(y'_{3,i}, Y3) = 0 then
        RemoveDUShoots(3, y'_{3,i})
      if (3, i) ∉ NewDUShootSet then
        CheckDUnaware(x_{3,i}, X3)
      Adapt(3, x_{3,i}, y_{3,i}, ⊥, ⊥)
      if ∃K ≠ K_1, K_2 : ETable[K]^{-1}(y_{3,i}) ∈ P_1 then
        Assert((1, K, x_{3,i}) ∈ Completed)
      UpdateCompleted(3, K_1, x_{3,i})
      CollectTP(3, x_{3,i}, y_{3,i})
    ProcessedShoots ← ProcessedShoots ∪ {(3, y_{3,i}, {K_1, K_2}), (3, y'_{3,i}, {K_1, K_2})}
    y'_{1,i+1} ← P_1(x'_{1,i+1}), y_{1,i+1} ← y'_{1,i+1} ⊕ k_1 ⊕ k_2
    if x_{1,i+1} ∈ P_1 or y_{1,i+1} ∈ P_1^{-1} then
      Assert(∃(1, x_{1,i+1}, {K, K'}) ∈ ProcessedShoots)
      Assert(P_1(x_{1,i+1}) = y_{1,i+1})
    else
      if DAwareness(x'_{1,i+1}, X1) = 0 then
        RemoveDUShoots(1, x'_{1,i+1})
      if (1, i + 1) ∉ NewDUShootSet then
        CheckDUnaware(y_{1,i+1}, Y1)
      Adapt(1, x_{1,i+1}, y_{1,i+1}, ⊥, ⊥)
      if ∃K ≠ K_1, K_2 : ETable[K](x_{1,i+1}) ∈ P_3^{-1} then
        Assert((1, k, x_{1,i+1}) ∈ Completed)
      UpdateCompleted(1, K_2, x_{1,i+1})
      CollectTP(1, x_{1,i+1}, y_{1,i+1})
    ProcessedShoots ← ProcessedShoots ∪ {(1, x_{1,i+1}, {K_1, K_2}), (1, x'_{1,i+1}, {K_1, K_2})}
  foreach (i, z) ∈ NewDUShootSet do
    if i = 1 then
      DUShoots ← {(1, {(x_{1,i}, y_{1,i}), (x'_{1,i}, y'_{1,i})})}
    else // i = 3
      DUShoots ← {(3, {(x_{3,i}, y_{3,i}), (x'_{3,i}, y'_{3,i})})}
  Border ← Border ∪ {x_{1,1}, x'_{1,1}, x_{1,2t+1}, x'_{1,2t+1}}
```

// The code for PROCESS33SHOOT(x_3, y_3, K_1, K_2) is similar to PROCESS11SHOOT by symmetry, thus omitted.

```
private procedure UpdateCompleted(i, K, x_i)
  if k ∉ HTable then abort
  k ← HTable(K)
  for j from i to 3 do
    if x_j ∉ P_j then abort
    y_j ← P_j(x_j)
  y_{i−1} ← x_i ⊕ k
  for j from i − 1 to 1 do
    if y_j ∉ P_j^{−1} then abort
    x_j ← P_j^{−1}(y_j)
  if ETable[K](x_1) ≠ y_3 then abort
  Completed ← Completed ∪ {(1, K, x_1), (2, K, x_2), (3, K, x_3)}
```

```
private procedure Process21TP(x_1, y_1, K)
  k ← HTable(K)
  Assert(∄k' : y_1 ⊕ k ⊕ k' ∈ P_1^{−1})
  x_2 ← y_1 ⊕ k, y_2 ← P_2(x_2), x_3 ← y_2 ⊕ k
  y_3 ← E.E(K, x_1)        [ y_3 ← EIn(K, x_1) ]
  CheckDUnaware(x_3, X3)
  Adapt(3, x_3, y_3, ⊥, ⊥)
  UpdateCompleted(3, K, x_3)
  // (3, x_3, y_3, ⊥) should not triggered new 13-TPs.
  Assert(∀K' ≠ K : ETable[K']^{−1}(y_3) ∉ P_1)
  CollectTP(3, x_3, y_3)
```

```
public procedure P1(x_1)
  CheckDUnaware(x_1, X1)
  return P1In(x_1)
```

```
private procedure P1In(x_1)
  if x_1 ∈ P_1 then return P_1(x_1)
  y_1 ← RandAssign(1, x_1, +)
  foreach (K, k, x_3, y_3) : (K, k) ∈ HQueries
          and (3, x_3, y_3) ∈ Queries do
    if Check(K, x_1, y_3) = true then
      x_2 ← y_1 ⊕ k, y_2 ← x_3 ⊕ k
      Take (K, x_1, y_3, edir, enum) from EQueries
      Adapt(2, x_2, y_2, edir, enum)
      UpdateCompleted(2, K, x_2)
      // No new 12-/32-TP.
      Assert(∄k' ≠ k : x_2 ⊕ k' ∈ P_1^{−1})
      Assert(∄k' ≠ k : y_2 ⊕ k' ∈ P_3)
  return P_1(x_1)
```

```
public procedure P2(x_2)
  CheckDUnaware(x_2, X2)
  if x_2 ∈ P_2 then
    y_2 ← P_2(x_2)
    foreach k ∈ Z do
      x_3 ← P_2(x_2) ⊕ k
      if x_3 ∈ P_3 then
        RemoveDUShoots(3, P_3(x_3))
    return P_2(x_2)
  Assert(♯{k|x_2 ⊕ k ∈ P_1^{−1}} ≤ 1)
  foreach (K, k) ∈ HQueries do
    if x_2 ⊕ k ∉ P_1^{−1} then continue
    x_1 ← P_1^{−1}(x_2 ⊕ k)
```

```
private procedure Process23TP(x_3, y_3, K)
  k ← HTable(K)
  Assert(∄k' : x_3 ⊕ k ⊕ k' ∈ P_3)
  y_2 ← x_3 ⊕ k, x_2 ← P_2^{−1}(y_2), y_1 ← x_2 ⊕ k
  x_1 ← E.E^{−1}(K, y_3)        [ x_1 ← EIn^{−1}(K, y_3) ]
  CheckDUnaware(y_1, Y1)
  Adapt(1, x_1, y_1, ⊥, ⊥)
  UpdateCompleted(1, K, x_1)
  // (1, x_1, y_1, ⊥) should not triggered new 31-TPs.
  Assert(∀K' ≠ K : ETable[K'](x_1) ∉ P_3^{−1})
  CollectTP(1, x_1, y_1)
```

```
public procedure P3^{−1}(y_3)
  CheckDUnaware(y_3, Y3)
  return P3In^{−1}(x_1)
```

```
private procedure P3In^{−1}(x_1)
  if y_3 ∈ P_3^{−1} then return P_3^{−1}(y_3)
  x_3 ← RandAssign(3, y_3, −)
  foreach (K, k, x_1, y_1) : (K, k) ∈ HQueries
          and (1, x_1, y_1) ∈ Queries do
    if Check(K, x_1, y_3) = true then
      x_2 ← y_1 ⊕ k, y_2 ← x_3 ⊕ k
      Take (K, x_1, y_3, edir, enum) from EQueries
      Adapt(2, x_2, y_2, edir, enum)
      UpdateCompleted(2, K, x_2)
      // No new 12-/32-TP.
      Assert(∄k' ≠ k : x_2 ⊕ k' ∈ P_1^{−1})
      Assert(∄k' ≠ k : y_2 ⊕ k' ∈ P_3)
  return P_3^{−1}(y_3)
```

```
    y_3 ← E.E(K, x_1)        [ y_3 ← EIn(K, x_1) ]
    P3In^{−1}(y_3)
  if x_2 ∉ P_2 then RandAssign(2, x_2, +)
  return P_2(x_2)
```

```
public procedure P2^{−1}(y_2)
  CheckDUnaware(y_2, Y2)
  if y_2 ∈ P_2^{−1} then
    x_2 ← P_2^{−1}(y_2)
    foreach k ∈ Z do
      y_1 ← P_2^{−1}(y_2) ⊕ k
      if y_1 ∈ P_1^{−1} then
        RemoveDUShoots(1, P_1^{−1}(y_1))
```

```
    return P_2^{-1}(y_2)
  Assert(♯{k|y_2 ⊕ k ∈ P_3} ≤ 1)
  foreach (K,k) ∈ HQueries do
    if y_2 ⊕ k ∉ P_3 then continue
    y_3 ← P_3(y_2 ⊕ k)

private procedure Adapt(i, x_i, y_i, ad2dir, ad2num)
  if x_i ∈ P_i or y_i ∈ P_i^{-1} then abort
  AddQuery(i, x_i, y_i, ⊥)
  if i = 2 then
    foreach k ∈ Z do
      AD2Edges ← AD2Edges ∪ {(x_2 ⊕ k, y_2 ⊕ k, ad2dir, ad2num)}
```

```
    x_1 ← E.E^{-1}(K, y_3)        [x_1 ← EIn^{-1}(K, y_3)]
    P1In(x_1)
  if y_2 ∉ P_2^{-1} then RandAssign(2, y_2, −)
  return P_2^{-1}(y_2)
```

6 Intermediate System G_2, and Stages of the Proof

To simplify the proof, we utilize an intermediate system denoted G_2. G_2 takes the same random tuples as G_1, but deviates from G_1 in the following aspects:

(i) The E-queries appear in the execution D^{G_2} are "explicitly bookkept" in a set *EQueries*, and the procedure Check(K, x_1, y_3) returns **true** only if (K, x_1, y_3) in is *EQueries*. This is similar to [CHK+16];

(ii) For the *meta-data transferring mechanism*, G_2 maintains a set *AD2Edges* for the edges formed by AD-2-queries (which were mentioned in Introduction);

(iii) G_2 have some abort conditions around the received queries, which capture the cases of D successfully guessing the history-values unknown to it. This is similar to [DRST12], while the design is much more complicated;

(iv) G_2 have some other "early abort" conditions, which capture the cases of G_2 obtaining bad random values. Roughly speaking, right after G_2 gets an n-bit random answer (may be x_i, y_i, or k) from either **E** or **R**, if the answer can be derived from the values in the history via certain relations, then G_2 aborts and would not add the query record containing this bad random value to the history. This is similar to [ABD+13a]. G_2's abortion due to these conditions will be referred to as *early-abortion*;

(v) In some cases, we expect certain properties hold, e.g. some new queries would not set off TPs/shoots, or some queries have been in history. In G_2, we use calls to a procedure Assert to ensure such expected properties: once they do not hold, the corresponding assertion fails, and G_2 aborts. We will show that if *early-abortion* never happens, then these assertions indeed never fail (jumping ahead, see Lemmata 12-15);

(vi) Finally, to simply some proof language, we let G_2 maintain an integer *cycleStartNum* for the *qnum* value of the most recent query from D, i.e. the "starting point" of the current chain-reaction.

We describe the first three (more complex) points in detail in the following subsubsections.

Bookkeeping E-queries, and the New Check. Initially, $EQueries = \emptyset$. Each new call to EIn(K, x_1) would gets $y_3 \leftarrow \mathbf{E}.\mathrm{E}(K, x_1)$ and adds a record $(K, x_1, y_3, \rightarrow, qnum)$ to *EQueries*, if early-abortion does not happen with respect to y_3. Here *EQueries* and the set *Queries*, *HQueries* share the same counter *qnum*. Symmetrically, each new call to EIn$^{-1}(K, y_3) \rightarrow x_1$ adds a record $(K, x_1, y_3, \leftarrow, qnum)$ to *EQueries* if early-abortion does not happen. In such cases we say G_2 *creates an E-query.*

As all the tuples in *EQueries* are consistent with an ideal cipher **E**, *EQueries* always defines a partial blockcipher. To simplify the language, we write $ETable[K]$ for $\{x_1 : \exists y_3, dir, num \text{ s.t. } (K, x_1, y_3, dir, num) \in EQueries\}$, and $ETable[K](x_1)$ for the corresponding y_3. Similarly for $ETable[K]^{-1}$ and $ETable[K]^{-1}(y_3)$. As mentioned, the procedure Check(K, x_1, y_3) in G_2 returns **true** only if $(K, x_1, y_3) \in EQueries$, in contrast to Check$(K, x_1, y_3)$ in G_1, which returns **true** whenever $\mathbf{E}.\mathrm{E}(K, x_1) = y_3$.

Meta-data Transferring Mechanism. *AD2Edges* is updated in two cases. First, each time G_2 creates an AD-2-query $(2, x_2, y_2, \perp)$, it finds the E-query $(K, x_1, y_3, edir, enum)$ corresponding to the chain under completion, and then adds a tuple $(x_2 \oplus k, y_2 \oplus k, k, edir, enum)$ to a set *AD2Edges* for each $k \in \mathcal{Z}$. Second, each time G_2 creates an H-query (K, k), for each AD-2-query $(2, x_2, y_2, \perp) \in Queries$ it picks the tuple $(x_2 \oplus k', y_2 \oplus k', k', edir', enum')$ from *AD2Edges* for an arbitrary $k' \in \mathcal{Z} \backslash \{k\}$ and then adds a new tuple $(x_2 \oplus k, y_2 \oplus k, k, edir', enum')$ to *AD2Edges*.

Accessing the meta-data of E-queries during adaptations is clearly an "illegal" operation for the real simulator S. But, G_2 is an imagined intermediate system, thus no problematic issues. One could also "manually" define such edges in the proof instead of letting G_2 explicitly maintain them. However, this seems more complicated.

Queries that Are Unaware to *D*. Recall from subsection 4.3 that *the goal is to prevent D from obtaining the values in shoots at the "endpoints" of the rhizomes.* Thus we only have to design a mechanism around the queries created in ProcessShoot-calls. Our solution is to take the shoots "internally" created in ProcessShoot-calls as "unknown" to D, and once D's query can be derived from the values in these shoots via certain relations, we let G_2 abort – this corresponds to G_2 succeeding in guessing a value relevant to these "unknown" shoots/values.

More clearly, we let G_2 maintain a set *Border* of n-bit values. The four values $x_{1,1}, x'_{1,1}, x_{1,2t+1}, x'_{1,2t+1}$ (in a Process11Shoot-call) or $y_{3,1}, y'_{3,1}, y_{3,2t+1}, y'_{3,2t+1}$ (in a Process33Shoot-call, cf. subsection 4.4) at the endpoints of the two alternated E-chains would be added to *Border*, to remind that they are "endpoints".

We let G_2 maintain another set *DUShoots* to keep the shoots that are supposed to be "fully unknown" to the distinguisher. The mechanism around this set is more sophisticated, and we divide it into the following three paragraphs: when to add new tuples to this set, how this set blocks D's aimlessly guessing, and when should G_2 remove tuples from this set.

Adding Tuples to *DUShoots*. In a call to Process11Shoot, the shoots with all the involved values newly obtained from the randomness source $(\mathbf{E}, \mathbf{R})$ would be added to *DUShoots*. E.g. for a 11-shoot formed by $(1, x_{1,i}, y_{1,i})$ and $(1, x'_{1,i}, y'_{1,i})$, if $x_{1,i}$, $y_{1,i}$, and $x'_{1,i}$ are all newly given by $(\mathbf{E}, \mathbf{R})$ in this call (on the other hand, $y'_{1,i}$ is necessarily derived from $y_{1,i}$), then a 2-tuple $(1, \{(x_{1,i}, y_{1,i}), (x'_{1,i}, y'_{1,i})\})$ is added to *DUShoots* – the second coordinate of this tuple is also a *set*. Symmetrically, 2-tuples of the form $(3, \{(x_{3,i}, y_{3,i}), (x'_{3,i}, y'_{3,i})\})$ are added to *DUShoots* for proper 33-shoots.

Similarly, in a a call to Process33Shoot, the "fresh" shoots have their records added to *DUShoots*.

Checking "D-awareness". To check whether D succeeds in guessing some history-values that are should have been unknown to it, each time G_2 receives a query from D, it makes a call to a procedure CheckDUnaware, which aborts depending on the situation.

Upon D querying $\mathrm{E}(K, x_1)$ or $\mathrm{P1}(x_1)$, if there exists a tuple of the form $(1, \{(x_1, P_1(x_1)), (\cdot, \cdot)\})$ in *DUShoots* – in this case we say *the 1-query* $(1, x_1, P_1(x_1))$ *is in DUShoots*, – G_2 aborts. Upon D querying $\mathrm{E}^{-1}(K, y_3)$ or $\mathrm{P3}^{-1}(y_3)$, G_2 checks symmetrically.

Upon D querying $\mathrm{P1}^{-1}(y_1)$, the conditions are more cumbersome: if there exists a tuple of the form $(1, \{(x'_1, y'_1), (x''_1, y''_1)\})$ in *DUShoots* such that $y_1 \oplus y'_1 \in 6\mathcal{Z}$ or $y_1 \oplus y''_1 \in 6\mathcal{Z}$, then G_2 aborts. The ideas are as follows:

(i) First, the value y_1 should not be unknown to D;
(ii) Second, the (possibly) newly created query $(1, x_1, y_1)$ should not form any TP nor shoot with the queries in *DUShoots*.

Upon D querying $\mathrm{P3}(x_3)$ the checking is similar by symmetry. Finally, upon D querying $\mathrm{P2}(x_2)$, if there exists a tuple of the form $(1, \{(x_1, y_1), (x'_1, y'_1)\})$ in *DUShoots* such that $x_2 \oplus y_1 \in 5\mathcal{Z}$ or $x_2 \oplus y'_1 \in 5\mathcal{Z}$, then G_2 aborts; symmetrically for D querying $\mathrm{P2}^{-1}(y_2)$.

Besides the above cases, internally created queries may have their values known to D. Such queries should not form any TP nor shoot with the queries in *DUShoots* either. Thus right before internally creating any query with values supposed to be known to D (e.g. Process21TP creating an AD-3-query), G_2 performs the same checking as above, as if this is a query newly received from D. And if the query does not pass the checking, G_2 aborts, thus avoiding creating this "bad" query.

One may notice that *we never check whether a queried main-key K is unknown to D or not.* This is not surprising: because all the internally obtained random values are n-bit ones, and G_1/G_2 *never* tries to use some main-key that is supposed to be unknown to D.

Removing Tuples from *DUShoots*. A tuples in *DUShoots* will be removed, once its "full unawareness" to D is supposed to be destroyed. It's performed by a procedure RemoveDUShoots, and is divided into two cases.

First, upon a query from D (that passes CheckDUnaware), the answer is clearly known to D, and the tuples in $DUShoots$ with values that can be derived from this answer via certain relations are removed from $DUShoots$. For example, when D queries $\mathrm{P2}^{-1}(y_2) \to x_2$, the tuples $(1, \{(x_1, y_1), (x'_1, y'_1)\})$ with $y_1 = x_2 \oplus k$ or $y'_1 = x_2 \oplus k$ for some $k \in \mathcal{Z}$ are removed.

Second, in a ProcessShoot-call, G_2 may internally "evaluates into" a shoot in $DUShoots$, and create some queries around it. In this case, the shoot may remain "fully unknown" to D, but its structure has been destroyed; thus in this case, we let G_2 remove this shoot from $DUShoots$ to keep clean structural properties for all the shoots in $DUShoots$.

6.1 Stages of the Proof

We consider a fixed, deterministic distinguisher D, and assume D issues q_e, q_h, and q_p queries to $\mathrm{E}/\mathrm{E}^{-1}$, H, and $\mathrm{P}i/\mathrm{P}i^{-1}$ respectively. The proof involves three systems as above: the *simulated* G_1, the *intermediate* G_2, and the *real* G_3. Note that $G_1(\mathbf{E}, \mathbf{R})$ and $G_2(\mathbf{E}, \mathbf{R})$ behave the same in the view of D, if:

- none of the additional abort-conditions in $G_2(\mathbf{E}, \mathbf{R})$ is fulfilled in $D^{G_2(\mathbf{E},\mathbf{R})}$, and
- the Check calls in $D^{G_1(\mathbf{E},\mathbf{R})}$ and $D^{G_2(\mathbf{E},\mathbf{R})}$ return the same answers.

On the other hand, via a randomness mapping, D^{G_2} and D^{G_3} could be related. Thus the crux of the proof is the analysis of D^{G_2}. In the remaining sections, first in Section 7, we collect some basic properties of D^{G_2}; then in Section 8, we prove adaptations and assertions would not cause D^{G_2} abort; thus we could give the simulator termination argument for G_2 in Section 9, and collect the probability of G_2 aborting in Section 10. Finally, in Section 11, we formally show G_1, G_2, and G_3 are indistinguishable, thus transiting the non-abortion and termination results in G_2 to G_1:

$$|Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} = 1] - Pr_{\mathbf{E},\mathbf{R}}[D^{G_1(\mathbf{E},\mathbf{R})} = 1]| \leq \frac{338 q_h (q_e + q_p)^2 \cdot q_p^4}{N}, \text{ (Lemma 20, subsection 11.1)}$$

$$|Pr_{\mathbf{R}}[D^{G_3(\mathbf{R})} = 1] - Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} = 1]| \text{ (Lemma 28, subsection 11.2)}$$
$$\leq \frac{2176 q_h^6 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{2359 q_e^2 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{338 q_e \cdot q_h (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{q_h^2 + q_h^4 + 8 q_e^2 + q_e \cdot q_h}{N}.$$

Note that subsection 11.2 achieves the goal via our partial-randomness-mapping argument. Gathering the above yields Theorem 1.

7 Basic Properties of G_2 Executions

This section presents some basic properties around G_2 executions.

7.1 Terminology, Helper Functions, and Equivalent Shoots

We first borrow the terminology *simulator cycle* from [ABD+13a], which refers to the execution period from the point D makes a query till the point D receives the answer – or G_2 aborts. Depending on the query of D, cycles are divided into three types:

- Cycles due to D querying E or E^{-1} are *E-cycles*;
- Cycles due to D querying $\mathrm{P}i$ or $\mathrm{P}i^{-1}$ are *P-cycles*;
- Cycles due to D querying H are *H-cycles*.

We further distinguish between *short simulator cycles* and *long simulator cycles*:

- Cycles induced by D querying P1, P2, $\mathrm{P2}^{-1}$, $\mathrm{P3}^{-1}$, and H are *short* ones. By the code, G_2 simply processes several 13-/31-TPs in such cycles;
- Cycles induced by D querying $\mathrm{P1}^{-1}$ and P3 are *long* ones. By the code, a lot of calls may emerge in such cycles, including Process11Shoot, Process21TP, etc. The analysis of such cycles would be the hardest point of our proof.

We then introduce two functions $xebval_l$ and $yebval_l$ to help probe in the (K, K')-alternated E-chains defined by $ETable$. Briefly speaking, $xebval_l$ takes two main-keys K and K' as well as a staring point x_1 as inputs, and moves in the (K, K')-alternated E-chain for l steps, and return the obtained new value y'_3 (in case l is odd) or x'_1 (in case l is even), or $\perp$, if the value is not computable due to the lack of some E-queries. $yebval_l$ is symmetrical to $xebval_l$. They are implemented as follows.

```
function xebval_l(K, K', x_1)
  j ← 0
  z ← x_1
  while j < l do
    if j is even then
      if z ∉ ETable[K] then return ⊥
      z ← ETable[K](z)
    else // j is odd
      if z ∉ ETable[K']^{-1} then return ⊥
      z ← ETable[K']^{-1}(z)
  return z
```

```
function yebval_l(K, K', y_3)
  j ← 0
  z ← y_3
  while j < l do
    if j is even then
      if z ∉ ETable[K]^{-1} then return ⊥
      z ← ETable[K]^{-1}(z)
    else // j is odd
      if z ∉ ETable[K'] then return ⊥
      z ← ETable[K'](z)
  return z
```

Based on these functions, we define *equivalent* shoots. Briefly speaking, shoots rooted at the same alternated E-chain with the same keys are equivalent; additionally, during G_2 processing a shoot, it would "reach" every shoots that are equivalent to this shoot.

Definition 2. *Two shoots* $(i, z, \{K_1, K_2\})$ *and* $(j, z', \{K_1, K_2\})$ *(with the same keys) are* equivalent *(denoted* $(i, z, \{K_1, K_2\}) \equiv (j, z', \{K_1, K_2\})$*), if:*

- $(i, z) = (j, z')$, *or*
- $z = xebval_l(K_1, K_2, z') \vee z = xebval_l(K_2, K_1, z')$ *(when* $j = 1$*), or*
- $z = yebval_l(K_1, K_2, z') \vee z = yebval_l(K_2, K_1, z')$ *(when* $j = 3$*).*

7.2 Invariants: for Structural Properties, and Chain-completion

Due to the incorporated early abort conditions (Section 6), certain features in *Queries*, *HQueries*, and *EQueries* are ensured at *any* point in *any* G_2 execution. First, each tuple in *Completed* corresponds to a completed path.

Lemma 1. *At any point in a* G_2 *execution, for any* $(i, K, x_i) \in Completed$*, there exist five queries in HQueries, Queries and EQueries as follows:*

$$(K, k), (K, x_1, y_3), (1, x_1, y_1), (2, x_2, y_2), (3, x_3, y_3), \text{ with } y_1 \oplus x_2 = y_2 \oplus x_3 = k.$$

Proof. *Completed* is fully maintained by UPDATECOMPLETED. By inspection of this procedure, it can be seen that only the tuples satisfying the requirements can be added to *Completed*, thus the claim. □

We then present several invariants, which are somewhat similar to [ABD+13a].

Inv1. (About the derived round-keys) There does not exist a pair of distinct main-keys K_1, K_2 such that $HTable(K_1) = HTable(K_2)$, nor four distinct $k_1, k_2, k_3, k_4 \in \mathcal{Z}$ such that $k_1 \oplus k_2 \oplus k_3 \oplus k_4 = 0$ (Similarly to [ABD+13a], this is ensured by H.).

Inv2. (About two P-queries to two consecutive rounds) For $n > n'$, there does not exist two queries $(i, x_i, y_i, \rightarrow, n)$ and $(i+1, x_{i+1}, y_{i+1}, d, n')$ (in *Queries*) such that $y_i \oplus x_{i+1} \in 5\mathcal{Z}$; there does not exist two queries $(i+1, x_{i+1}, y_{i+1}, \leftarrow, n)$ and (i, x_i, y_i, d, n') such that $y_i \oplus x_{i+1} \in 5\mathcal{Z}$ either. (This is ensured by ADDQUERY and H. Jumping ahead, the full power of this Inv is used in Lemma 6 and Proposition 8.)

Inv3. (About two P-queries to the same round) For $n > n'$, there does not exist two queries $(i, x_i, y_i, \rightarrow, n)$ and (i, x'_i, y'_i, d, n') such that $y_i \oplus y'_i \in 6\mathcal{Z}$; there does not exist two queries $(i, x_i, y_i, \leftarrow, n)$ and (i, x'_i, y'_i, d, n') such that $x_i \oplus x'_i \in 6\mathcal{Z}$ (ensured by ADDQUERY and H, with full power used in Lemma 6 and Proposition 8.).

Inv4. (About two E-queries) For $n > n'$, there does not exist two E-queries $(K, x_1, y_3, \rightarrow, n)$ and (K', x'_1, y_3, d, n'); there does not exist two E-queries $(K, x_1, y_3, \leftarrow, n)$ and (K', x_1, y'_3, d, n') (ensured by EIN and EIN^{-1}).

Inv5. (About an E-query and a 1/3-query) Directed 1/3-queries and E-queries never head towards each other (obviously follows from ADDQUERY, EIN, and EIN^{-1}):

- There does not exist an E-query (K, x_1, y_3, d_e, n_e) and a 1-query $(1, x_1, y_1, d_1, n_1)$ such that either: (i) $n_1 > n_e$ and $d_1 =\leftarrow$, or (ii) $n_e > n_1$ and $d_e =\leftarrow$;
- There does not exist an E-query (K, x_1, y_3, d_e, n_e) and a 3-query $(3, x_3, y_3, d_3, n_3)$ such that either: (i) $n_3 > n_e$ and $d_3 =\rightarrow$, or (ii) $n_e > n_3$ and $d_e =\rightarrow$.

It's not hard to see that the above five invariants hold throughout any G_2 execution.

The remaining three Invs state that the tripwires and rhizome-mechanism function as wished.

Inv6. ("Static" TPs indicate completed paths) In each of the following cases, the involved queries are part of the same K-completed path, and the 3-tuples corresponding to the path are in *Completed*:

(i) There are three queries (K, k), (i, x_i, y_i), and $(i+1, x_{i+1}, y_{i+1})$ $(i = 1, 2)$ such that $k = y_i \oplus x_{i+1}$;
(ii) There are three queries (K, k), $(1, x_1, y_1)$, and $(3, x_3, y_3)$ such that $G_2.\textsc{Check}(K, x_1, y_3) = \textbf{true}$.

Inv7. (Processed shoots indicate completed paths) For any tuple $(1, x_1, \{K_1, K_2\}) \in \mathit{ProcessedShoots}$, let $x_1' = P_1^{-1}(P_1(x_1) \oplus k_1 \oplus k_2)$. If $x_1 \notin \mathit{Border}$, then $(1, K_1, x_1)$, $(1, K_2, x_1)$, $(1, K_1, x_1')$, and $(1, K_2, x_1')$ are all in *Completed*; otherwise, for $x \in \{x_1, x_1'\}$ and $(K, k) \in \{(K_1, k_1), (K_2, k_2)\}$, the tuples $(1, K, x)$ such that $P_1(x) \oplus k \in P_2$ are in *Completed.*

Symmetrically, for any tuple $(3, y_3, \{K_1, K_2\}) \in \mathit{ProcessedShoots}$, let $x_3 = P_3^{-1}(y_3)$, and $x_3' = x_3 \oplus k_1 \oplus k_2$. If $y_3 \notin \mathit{Border}$, then $(3, K_1, x_3)$, $(3, K_2, x_3)$, $(3, K_1, x_3')$, and $(3, K_2, x_3')$ are all in *Completed*; otherwise, for $x \in \{x_3, x_3'\}$ and $(K, k) \in \{(K_1, k_1), (K_2, k_2)\}$, the tuples $(3, K, x)$ such that $x \oplus k \in P_3^{-1}$ are in *Completed.*

Inv8. ("Static" shoots indicate processed shoots) For any two 1-queries $(1, x_1, y_1)$ and $(1, x_1', y_1')$ such that $y_1 \oplus y_1' = k_1 \oplus k_2$ for some H-queries (K_1, k_1) and (K_2, k_2), both $(1, x_1, \{K_1, K_2\})$ and $(1, x_1', \{K_1, K_2\})$ are in *ProcessedShoots*. Moreover, any $(i, z, \{K_1, K_2\}) \equiv (1, x_1, \{K_1, K_2\})$ and $(i, z', \{K_1, K_2\}) \equiv (1, x_1', \{K_1, K_2\})$ are also in *ProcessedShoots*.

Symmetrically, for any two 3-queries $(3, x_3, y_3)$ and $(3, x_3', y_3')$ such that $x_3 \oplus x_3' = k_1 \oplus k_2$ for some H-queries (K_1, k_1) and (K_2, k_2), both $(3, y_3, \{K_1, K_2\})$ and $(3, y_3', \{K_1, K_2\})$ are in *ProcessedShoots*. Moreover, any $(i, z, \{K_1, K_2\}) \equiv (3, y_3, \{K_1, K_2\})$ and $(i, z', \{K_1, K_2\}) \equiv (3, y_3', \{K_1, K_2\})$ are also in *ProcessedShoots*.

Lemma 2. *Inv6-Inv8 hold at the end of each simulator cycle as long as G_2 does not abort.*

Proof. We prove Inv7 first. Note that tuples of the form $(1, x_1, \{K_1, K_2\})$ can only be added to *ProcessedShoots* in ProcessShoot-calls. The claim thus can be seen from the code of ProcessShoot.

We then consider Inv8. Wlog consider four queries $(1, x_1, y_1, d_1, n_1)$, $(1, x_1', y_1', d_1', n_1')$, (K_1, k_1, nk_1), and (K_2, k_2, nk_2) with $y_1 \oplus y_1' = k_1 \oplus k_2$. It must be $nk_1, nk_2 \neq \mathrm{Max}\{n_1, n_1', nk_1, nk_2\}$, otherwise G_2 would have aborted in $\mathrm{H}(K_1)$ or $\mathrm{H}(K_2)$ and not create (K_1, k_1) nor (K_2, k_2). Thus wlog assume $n_1 = \mathrm{Max}\{n_1, n_1', nk_1, nk_2\}$. Then $d_1 = \leftarrow$ or $\perp$ by Inv3, and we have two possibilities:

Case 1.1: $d_1 = \leftarrow$. By the code, $(1, x_1, y_1, \leftarrow)$ can only be created in $\mathrm{P1}^{-1}(y_1)$, after which $(1, x_1, \{K_1, K_2\})$ will be in *ShootQueue*. Later when $(1, x_1, \{K_1, K_2\})$ is popped, either $(1, x_1, \{K_1, K_2\}) \in \mathit{ProcessedShoot}$, or G_2 would call Process11Shoot(x_1, y_1, K_1, K_2), and if this call returns without abortion then the claim holds by the code.

Case 1.2: $d_1 = \perp$. It falls into three cases:

(i) $(1, x_1, y_1)$ is created in a call to Process23TP. Then by the code, the subsequent call to CollectTP$(1, x_1, y_1)$ would push $(1, x_1, \{K_1, K_2\})$ into *ShootQueue*, and the claim would hold after $(1, x_1, \{K_1, K_2\})$ is later popped without abortion;
(ii) $(1, x_1, y_1)$ is created in a call to Process11Shoot(x_1'', y_1'', K, K') or Process33Shoot(x_3'', y_3'', K, K') with $\{K, K'\} \neq \{K_1, K_2\}$. Then it's similar to the previous case: $(1, x_1, \{K_1, K_2\})$ would be pushed into *ShootQueue* and later popped and thus the claim;
(iii) $(1, x_1, y_1)$ is created in a call to Process11Shoot(x_1'', y_1'', K_1, K_2) or Process33Shoot(x_3'', y_3'', K_1, K_2). Wlog we focus on the former. Note that in this case, it necessarily be $(1, x_1, \{K_1, K_2\}) \equiv (1, x_1'', \{K_1, K_2\})$ or $(1, x_1', \{K_1, K_2\}) \equiv (1, x_1'', \{K_1, K_2\})$. Thus $(1, x_1, \{K_1, K_2\}), (1, x_1', \{K_1, K_2\}) \in \mathit{ProcessedShoots}$ holds after this call (once non-aborting).

We finally turn to Inv6, and consider three queries (K, k, n_k), $(1, x_1, y_1, d_1, n_1)$, and $(2, x_2, y_2, d_2, n_2)$ with $k = x_1 \oplus y_2$ first. Note that $n_k \neq \mathrm{Max}\{n_k, n_1, n_2\}$, otherwise G_2 would have aborted in $\mathrm{H}(K)$. Thus we have two possibilities:

Case 2.1: $n_1 = Max\{n_k, n_1, n_2\}$. Then $d_1 = \leftarrow$ or $\perp$, otherwise contradicting Inv2. According to the pseudocode, right after G_2 creating $(1, x_1, y_1, \leftarrow)$ or $(1, x_1, y_1, \perp)$, G_2 would make a call to CollectTP$(1, x_1, y_1)$. By the code of CollectTP, it falls into two cases:

(i) $\forall k' \in \mathcal{Z}\backslash\{k\}, y_1 \oplus k \oplus k' \notin P_1^{-1}$. Then a 21-TP $(1, x_1, K)$ is pushed into $MidTPQueue$. Thus when G_2 pops $(1, x_1, K)$, either $(1, K, x_1) \in Completed$, or G_2 calls Process21TP, after which $(1, K, x_1) \in Completed$ holds (once non-aborting);

(ii) $\exists(K', k') \in HQueries\backslash\{(K, k)\} : y_1 \oplus k \oplus k' \in P_1^{-1}$. Then G_2 detects a 11-shoot $(1, x_1, \{K, K'\})$, and:
  - if $(1, x_1, \{K, K'\}) \in ProcessedShoot$, then $(1, K, x_1) \in Completed$ by Inv7 (note that the existence of $(2, x_2, y_2)$ indicates $y_1 \oplus k \in P_2$);
  - if $(1, x_1, \{K, K'\}) \notin ProcessedShoot$, then $(1, x_1, \{K, K'\})$ is pushed into $ShootQueue$. Therefore, after being popped without abortion, $(1, x_1, \{K, K'\})$ is in $ProcessedShoots$. This further implies $(1, K, x_1) \in Completed$ by Inv7.

Case 2.2: $n_2 = Max\{n_k, n_1, n_2\}$. By Inv2 we get $d_2 = \rightarrow$ or $\perp$. According to the code, RandAssign$(2, x_2, +)$ only happens in P2, when $\nexists k \in \mathcal{Z} : x_2 \oplus k \in P_1^{-1}$. Thus d_2 necessarily equals $\perp$. According to the code around calls to Adapt$(2, x_2, y_2, \cdot, \cdot)$, one can see that right after G_2 creates $(2, x_2, y_2, \perp)$, it falls into one of the following three cases:

(i) $(2, K, x_2) \in Completed$ (Indeed, G_2 creating $(2, x_2, y_2, \perp)$ is exactly the adaptation of the chain $(2, K, x_2)$.), or

(ii) the purported 1-query $(1, x_1, y_1)$ dose not exist, or

(iii) $(1, x_1, \{K, K'\}) \in ShootQueue$ for some $K' \neq K$, thus by Inv7, the claims will hold after the cycle $((1, x_1, \{K, K'\})$ has been popped without abortion). The AD-2-queries created in the *Shoot-Growing-Phase* of ProcessShoot-calls may fall into this case.

The case of three queries (K, k), $(2, x_2, y_2)$, and $(3, x_3, y_3)$ with $k = x_2 \oplus y_3$ is similar by symmetry.

Then, consider three queries (K, k, n_k), $(1, x_1, y_1, d_1, n_1)$, and $(3, x_3, y_3, d_3, n_3)$. We also have two possibilities:

Case 3.1: $n_1 = Max\{n_k, n_1, n_3\}$. Then $d_1 \neq \leftarrow$ by Inv5. According to the statements subsequent to the call Adapt$(1, x_1, y_1, \perp, \perp)$, if $d_1 = \perp$, then either $(1, K, x_1) \in Complete$, or G_2 aborts – for example, if $(1, x_1, y_1)$ is created in a call to Process23TP(x_3, y_3, K'), then either $K' = K$, or the purported 3-query $(3, x_3, y_3)$ should not exist. On the other hand, if $d_1 = \rightarrow$, then $(1, x_1, y_1)$ is created in a call to P1In(x_1), or ProcessShoot with associated keys K_1, K_2. In the former case, according to the statements in P1In(x_1), after G_2 creating $(1, x_1, y_1)$, G_2 would find $(3, x_3, y_3)$ via calling Check, and thus $(1, K, x_1) \in Completed$ after this cycle once non-aborting. In the later case, if $K \neq K_1, K_2$, then it has no difference; if $K = K_1$ or K_2, then $(1, K, x_1) \in Completed$ also holds after this ProcessShoot-call returns.

The case of $n_3 = \text{Max}\{n_k, n_1, n_3\}$ is similar to the above case by symmetry.

Case 3.2: $n_k = Max\{n_k, n_1, n_3\}$. Then in the call to H, G_2 would find $(1, x_1, y_1)$ and $(3, x_3, y_3)$ via Check, and thus $(1, K, x_1) \in Completed$ after this cycle. □

7.3 Bipartite Graphs B_2, EB

This subsection presents formal definitions for the two bipartite graph B_2 and EB which encode the information from $Queries$ and $CQueries$, as well as discussions on their structural properties. Both B_2 and EB have shores $\{0, 1\}^n$, and are time-dependent.

We describe B_2 first. Edges of B_2 are directed and labeled, and constructed as follows. For every 2-query $(2, x_2, y_2, dir, num) \in Queries$ with $dir \neq \perp$ and every $k \in \mathcal{Z}$, we construct an RA-2-edge (y_1, x_3) of label k, of direction and an associated num value equaling the dir and num value of the 2-query respectively (this edge is "RA" because $(2, x_2, y_2)$ is created by RandAssign). For convenience, we use a 5-tuple $(y_1, x_3, k, edir, enum)$ to refer to this edge. For every 2-query $(2, x_2, y_2, \perp, num) \in Queries$ and every $k \in \mathcal{Z}$ we construct an AD-2-edge (y_1, x_3) labeled k, but the direction and edge-number do not follow the 2-query. Indeed, from the pseudocode one can see that for each such pair $(2, x_2, y_2, \perp, num)$ and k there would be a 5-tuple $(y_1, x_3, k, edir, enum)$ in $AD2Edge$; we take this tuple as the constructed edge. The above constitute all edges of B_2. Thus each edge of B_2 is associated to a pair comprised of one 2-query and of one H-query. We call 1-queries with $dir = \rightarrow$ and 3-queries with $dir = \leftarrow$ *heading towards* B_2.

We write $B_2(z)$ for the connected component in B_2 containing the vertex z. Note that $B_2(z)$ may contain only one node (say, the case of z not adjacent to any edge). Also note that $B_2(z)$ and $B_2(z')$ may be the same connected component even if $z \neq z'$; more clearly, one can see they are the same structure if and only if z is a node in $B_2(z')$ (or vice versa). In this case, we write $z \in B_2(z')$.

In B_2, a node y_1 in the left shore that satisfies $y_1 \in P_1^{-1}$ is called *pebbled*; symmetrically, a node x_3 with $x_3 \in P_3$ is *pebbled*.

One may see proving B_2 to be acyclic is *indispensable* for the proof: if the distinguisher is able to "create" a cycle structure in B_2, then the burden of finding a similar cycle of E-queries would be put on the simulator's shoulders, and this clearly could collapse any polynomial-complexity simulator. However this is not an easy task, and took us a lot of efforts. Due to its complexity, we defer this discussion to subsection 7.6. In a departure from this paper, the graph B_2 used in [ABD+13a], is easily seen to contains no multiple edges, since ABDMS's simulated 2-queries are always created by random assignments.

To help probe in B_2, we use two additional functions $yb2val_l$ and $xb2val_l$. They take two round-keys k and k' and a starting point as inputs and move in B_2 in a "(k, k')-alternated manner" (somewhat symmetrically to $xebval_l$ and $yebval_l$).

```
function yb2val_l(k, k', y_1)
  j ← 0
  z ← y_1
  while j < l do
    if j is even then
      if k ⊕ z ∉ P_2 then return ⊥
      z ← k ⊕ P_2(k ⊕ z)
    else // j is odd
      if k' ⊕ z ∉ P_2^{-1} then return ⊥
      z ← k' ⊕ P_2^{-1}(k' ⊕ z)
  return z
```

```
function xb2val_l(k, k', x_3)
  j ← 0
  z ← x_3
  while j < l do
    if j is even then
      if k ⊕ z ∉ P_2^{-1} then return ⊥
      z ← k ⊕ P_2^{-1}(k ⊕ z)
    else // j is odd
      if k' ⊕ z ∉ P_2 then return ⊥
      z ← k' ⊕ P_2(k' ⊕ z)
  return z
```

We then describe EB. For every E-query $(K, x_1, y_3, dir, num) \in EQueries$, we construct an edge (x_1, y_3) of label K, of direction dir ($dir \in \{\rightarrow, \leftarrow\}$ for E-queries), and of an associated num value equaling the num value of the E-query. This constitutes all edges of EB. We simply use the E-query (K, x_1, y_3, dir, num) to refer to the corresponding edge. Due to Inv4, two distinct E-queries cannot give rise to two edges of EB with the same endpoints, and thus EB contains no multiple edges. If (K, x_1, y_3) has been in a K-completed path, then we say the E-query/edge is *dead*, otherwise *live*.

We write $EB(z)$ for the connected component in EB containing the vertex z. Also, $EB(z)$ may contain only one node; and $EB(z)$ and $EB(z')$ are the same connected component if and only if z is a node in $EB(z')$ (denoted $z \in EB(z')$; or vice versa).

It's not hard to see that *for any z, $EB(z)$ is a tree*. The formal proof is almost the same as Lemmata 12 and 14 of [ABD+13a].

Proposition 1. *Connected components of EB are directed trees with edges directed away from the root, and the num values on the edges of any directed path in EB are strictly increasing.*

Proof. Due to Inv4, every vertex of EB has indegree at most 1. Moreover, since queries are totally ordered and a single E-query exactly raises a single edge in EB, two adjacent edges have different num values. Due to Inv4, these num values go from smaller to larger according to the edge directions, hence the connected component is also acyclic. ◻

Some of the trees in EB are more interesting than the others; to identify them, we follow [HKT11] and define *table-defined trees.*

Definition 3. *The tree $EB(x_1)$ is* table-defined*, if $x_1 \in P_1$ or $\exists K : x_1 \in ETable[K]$. Symmetrically, $EB(y_3)$ is* table-defined *if $y_3 \in P_3^{-1}$ or $\exists K : y_3 \in ETable[K]^{-1}$.*

Two different table-defined trees in EB never subsequently merge.

Proposition 2. *If both $EB(z)$ and $EB(z')$ are table-defined and $z \notin EB(z')$, then $z \in EB(z')$ is never possible.*

Proof. Consider two such trees $EB(z)$ and $EB(z')$. Wlog assume that a forward E-query $\mathrm{E}(K, x_1)$ such that $x_1 \in EB(z)$ appears, and this leads to G_2 creating $(K, x_1, y_3, \rightarrow, n_e)$ for $y_3 = \mathbf{E}.\mathrm{E}(K, x_1)$. Clearly n_e is larger than the num of any edge in $EB(z')$. Thus if y_3 falls into the edges of $EB(z')$, then it contradicts Inv4. On the other hand, if $EB(z')$ only contains z', then it has to be $y_3 = z'$; as $EB(z')$ was table-defined, $y_3 \in P_3^{-1}$ already held before $\mathrm{E}(K, x_1)$ appears, and it contradicts Inv5. Thus the claim. ◻

7.4 Internally Created E-queries Are Killed At Once

During chain-reaction calls (cf. subsection 5.1), G_2 may internally calls $\textsc{Ein}$ and $\textsc{Ein}^{-1}$, leading to creating new E-queries. However, these queries are killed soon.

Lemma 3. *Assume that G_2 processes a chain-reaction call without abortion. Then all the E-queries newly created in this call are dead right after this call is finished.*

Proof. By tracking the boxed statements in the pseudocode, the following six calls are able to lead to G_2 "internally" creating E-queries:

(i) $\textsc{Process21TP}$ and $\textsc{Process23TP}$;
(ii) $\textsc{Process11Shoot}$ and $\textsc{Process33Shoot}$;
(iii) P2 and P2^{-1}.

We then proceed to argue for each of the above calls:

For $\textsc{Process21TP}$ *and* $\textsc{Process23TP}$*:* Wlog consider a call to $\textsc{Process21TP}(x_1^\circ, y_1^\circ, K^\circ)$. By the code, this call first makes a query to $\textsc{Ein}(K, x_1^\circ)$ to obtain y_3°, and then obtains the value x_3° by accessing the sets. We distinguish two possibilities:

- right before the query to $\textsc{Ein}(K, x_1^\circ)$, it holds $x_1^\circ \notin ETable[K]$. Then a new E-query $(K, x_1^\circ, y_3^\circ, \rightarrow)$ is created. Note that at this point, by Inv4 it holds $\forall K' \neq K, y_3^\circ \notin ETable[K']^{-1}$. Thus only one E-query is created. Later after $\textsc{Adapt}(3, x_3^\circ, y_3^\circ)$ returns, this new query is in a completed path and thus dead;
- opposite to the first case – then no new E-query is created and the claim trivially follows.

Thus the claim holds for E-queries newly created in $\textsc{Process21TP}$- and $\textsc{Process23TP}$-calls.

For $\textsc{Process11Shoot}$ *and* $\textsc{Process33Shoot}$*:* Wlog consider a call to $\textsc{Process11Shoot}(x_1^\circ, y_1^\circ, K_1, K_2)$. Let $k_i = HTable(K_i)$ for $i = 1, 2$, and let $y_1^{\circ\circ} = y_1^\circ \oplus k_1 \oplus k_2$ and $x_1^{\circ\circ} = P_1^{-1}(y_1^\circ)$. For this lemma, we do not need to dig out all the details in $\textsc{Process11Shoot}$; instead, we simply note that the call would first take $(x_1^\circ, x_1^{\circ\circ})$ as $(x_{1,t+1}^\circ, x_{1,t+1}^{\circ\circ})$ and make the following two chains of E-queries

$$x_{1,1}^{\circ\circ} \xleftarrow{\textsc{Ein}_{K_2}^{-1}} \ldots \xleftarrow{\textsc{Ein}_{K_1}} x_{1,t}^{\circ\circ} \xleftarrow{\textsc{Ein}_{K_2}^{-1}} y_{3,t}^{\circ\circ} \xleftarrow{\textsc{Ein}_{K_1}} x_{1,t+1}^{\circ\circ} \xrightarrow{\textsc{Ein}_{K_2}} y_{3,t+1}^{\circ\circ} \xrightarrow{\textsc{Ein}_{K_1}^{-1}} x_{1,t+2}^{\circ\circ} \xrightarrow{\textsc{Ein}_{K_2}} \ldots \xrightarrow{\textsc{Ein}_{K_1}^{-1}} x_{1,2t+1}^{\circ\circ}$$

and

$$x_{1,1}^{\circ} \xleftarrow{\textsc{Ein}_{K_1}^{-1}} \ldots \xleftarrow{\textsc{Ein}_{K_2}} x_{1,t}^{\circ} \xleftarrow{\textsc{Ein}_{K_1}^{-1}} y_{3,t}^{\circ} \xleftarrow{\textsc{Ein}_{K_2}} x_{1,t+1}^{\circ} \xrightarrow{\textsc{Ein}_{K_1}} y_{3,t+1}^{\circ} \xrightarrow{\textsc{Ein}_{K_2}^{-1}} x_{1,t+2}^{\circ} \xrightarrow{\textsc{Ein}_{K_1}} \ldots \xrightarrow{\textsc{Ein}_{K_2}^{-1}} x_{1,2t+1}^{\circ}.$$

The claim could be established similarly to the argument for $\textsc{Process21TP}$: G_2 creates at most $8t$ new E-queries. Then in the *Fill-in-Rung-Phase*, $\textsc{Process11Shoot}$ would create a series of AD-2-queries. It can be seen from the calls to $\textsc{UpdateCompleted}$ that if G_2 does not abort, then the (at most $4t$) newly created E-queries adjacent to $x_1^{\circ\circ}$ are killed in this phase. Later in the *Shoot-Completing-Phase*, the $\textsc{Process11Shoot}$-call would attach an AD-1-query to each $x_{1,i}^\circ$ and an AD-3-query to each $y_{3,i}^\circ$. This however kills all the newly created E-queries adjacent to x_1° (also at most $4t$). Thus the claim holds for E-queries created in $\textsc{ProcessShoot}$-calls.

For P2 *and* P2^{-1}*:* Wlog consider $\text{P2}^{-1}(y_2)$ with $y_2 \notin P_2^{-1}$ (otherwise G_2 simply reads the records and does not call $\textsc{Ein}^{-1}$). G_2 first checks an assertion. If the assertion does not cause abort, then there exists exactly one (K, k) such that $x_3 = y_2 \oplus k \in P_3$. Let the involved 3-query be $(3, x_3, y_3)$, then we distinguish two possibilities:

(i) first, $y_3 \notin ETable[K]^{-1}$. Then the call to $\textsc{Ein}^{-1}(K, y_3)$ would lead to creating a new E-query $(K, x_1, y_3, \leftarrow)$. By Inv4 and Inv5, right after this point, it holds: (i) $\forall K' \neq K, x_1 \notin ETable[K']$; (ii) $x_1 \notin P_1$. By this, the subsequent call to $\text{P1}\textsc{In}(x_1)$ would lead to creating $(1, x_1, y_1, \rightarrow)$ with $y_1 = \mathbf{R}.\text{P1}(x_1)$ and G_2 completing the chain formed by $(1, x_1, y_1)$, (K, x_1, y_3), and $(3, x_3, y_3)$. It's clear that if this process is finished without abortion, then (K, x_1, y_3) would be in a complete path;
(ii) second, $y_3 \in ETable[K]^{-1}$. In this case G_2 does not create new E-queries.

Thus the claim holds for E-queries created in P2 and P2^{-1}. These complete the proof. ☐

As a corollary, 13- and 31-TPs are strongly relevant to D querying E or E^{-1}.

Proposition 3. *For any 13-/31-/H-TP, the involved E-query was necessarily created due to D querying* E *or* E^{-1}.

Proof. Right after a 13-, 31-, or H-TP is detected, the involved E-query is necessarily live. But by Lemma 3, any internally-created E-query would be dead after the call during which the query is created. Thus the claim. ☐

7.5 Properties of AD-1- and AD-3-queries

Note that AD-1- and AD-3-queries can only be created during long simulator cycles (this can be easily seen from the code or the overview of simulation strategy). The whole process of a long cycle could be informally described as follows. First, if a query $\mathrm{P1}^{-1}(z)$ or $\mathrm{P3}(z)$ sets off several tripwires, then the E-queries internally created by G_2 would form a tree in EB. This tree is "new", in the sense that it would not be adjacent to *any* table-defined trees that already existed in EB before the simulator cycle. Moreover, *all* the newly created AD-1- and AD-3-queries are attached to this tree.

On the other hand, during the cycle, G_2 also extends the connected component $B_2(z)$. More importantly, *all* the newly created AD-1- and AD-3-queries are also adjacent to $B_2(z)$.

Lemma 4. *Assume that D issues a new query $\mathrm{P1}^{-1}(z)$ or $\mathrm{P3}(z)$, which results in* RandAssign *returning z'. Then after this point, with respect to the connected components $EB(z')$ and $B_2(z)$, we have:*

(i) z' is the root of $EB(z')$. Moreover, for any tree $EB(z^*)$ that has been table-defined before the query $\mathrm{P1}^{-1}(z)$ (or $\mathrm{P3}(z)$, resp.), $z' \in EB(z^*)$ is never possible;

(ii) during the subsequent simulator cycle, all the newly created AD-1- and AD-3-queries are adjacent to both $EB(z')$ and $B_2(z)$;

(iii) after the subsequent simulator cycle, if G_2 does not abort, then all the E-queries in $EB(z')$ are dead.

Proof. We focus on a new $\mathrm{P1}^{-1}(y_1)$; the case of $\mathrm{P3}(x_3)$ is indeed similar. Assume $\textsc{RandAssign}(1, y_1, -)$ returns x_1. Then right after this point, we have $\forall K : x_1 \notin ETable[K]$ by Inv5. This means x_1 is not adjacent to any edge in EB. Thus all the paths in $EB(x_1)$ are necessarily directed away from x_1, i.e. x_1 is the root. Moreover, for any $EB(z^*)$ that has been table-defined before the query, $x_1 \in EB(z^*)$ does not hold at this point, and would never be possible by Lemma 1. Thus (i).

To show (ii), we show the following sub-claims:

- *Sub-claim 1:* for any call to $\textsc{CollectTP}(1, z, z', -)$ or $\textsc{CollectTP}(3, z', z, +)$, (a) if $z \in EB(x_1)$, then the root of each shoot and MidTP detected in this call lies in $EB(x_1)$; (b) if $z' \in B_2(y_1)$, then the peak of each shoot and MidTP detected in this call lies in $B_2(y_1)$;
- *Sub-claim 2:* for any MidTP to be processed by a call to Process21TP or Process23TP, if its root lies in $EB(x_1)$ and its peak lies in $B_2(y_1)$, then unless abortion occurs, (a) the AD-query created in this call is adjacent to both $EB(x_1)$ and $B_2(y_1)$; (b) the sub-call to CollectTP meets the requirement of *Sub-claim 1*;
- *Sub-claim 3:* for any call to Process11Shoot or Process33Shoot, if the root of the shoot to be processed lies in $EB(x_1)$ while the peak lies in $B_2(y_1)$, then unless abortion occurs, (a) the AD-1- and AD-3-queries created in this call are adjacent to both $EB(x_1)$ and $B_2(y_1)$; (b) the sub-calls to CollectTP meet the requirement of *Sub-claim 1*.

By these, (ii) can be proved via induction. We then argue for them one-by-one.

For sub-claim 1: Wlog we consider a call to $\textsc{CollectTP}(1, x_1^\circ, y_1^\circ, -)$; the argument for $\textsc{CollectTP}(3, x_3^\circ, y_3^\circ, +)$ has no essential difference. This CollectTP-call would check the entries in sets and push the newly detected shoots and 21-TPs into *ShootQueue* and *MidTPQueue* respectively. It can be seen from the code that: all the newly enqueued shoots are of the form $(1, x_1^\circ, \{K^\circ, K^{\circ\circ}\})$ for some $K^\circ, K^{\circ\circ}$, the root of which is $x_1^\circ \in EB(x_1)$, and the peak is $y_1^\circ \in B_2(y_1)$; all the newly enqueued 21-TPs are of the form $(1, x_1^\circ, K^*)$ for some K^*, which is also rooted at x_1° and "peaked" at y_1°. Thus the (sub-)claim.

For sub-claim 2: Wlog we consider a call to $\textsc{Process21TP}(x_1^\circ, y_1^\circ, K^\circ)$. Recall from Lemma 3 that this call first makes a query to $\textsc{EIn}(K, x_1^\circ)$ to obtain y_3°, and then obtains the value x_3° by accessing the sets. If abortion does not occur, then $y_3^\circ \in EB(x_1)$ clearly holds; and $x_3^\circ \in B_2(y_1)$ as $x_3^\circ = k^\circ \oplus P_2(k^\circ \oplus y_1^\circ)$ and $y_1^\circ \in B_2(y_1)$. Thus the AD-3-query created by the sub-call to $\textsc{Adapt}(3, x_3^\circ, y_3^\circ)$ is adjacent to $EB(x_1)$ and $B_2(y_1)$. Moreover, the subsequent call is to $\textsc{CollectTP}(3, x_3^\circ, y_3^\circ, +)$, which clearly meets the requirement of *Sub-claim 1* (i.e. $y_3^\circ \in EB(x_1)$ and $x_3^\circ \in B_2(y_1)$).

For sub-claim 3: Wlog consider a call to Process11Shoot$(x_1^\circ, y_1^\circ, K_1, K_2)$, and let $y_1^{\circ\circ} = y_1^\circ \oplus k_1 \oplus k_2$ and $x_1^{\circ\circ} = P_1^{-1}(y_1^\circ)$. Following the flow analyzed in Lemma 3, we note that the Process11Shoot-call would first take $(x_1^\circ, x_1^{\circ\circ})$ as $(x_{1,t+1}^\circ, x_{1,t+1}^{\circ\circ})$ and make two chains of E-queries, each with length $4t$. Cf. the proof of Lemma 3 for an illustration of these two chains, which are omitted here to save space.

Then in the *Fill-in-Rung-Phase*, Process11Shoot would create a series of AD-2-queries. It can be seen that these AD-2-queries form a (k_1, k_2)-alternated path in B_2 with length $4t$, which is adjacent to y_1.

Later in the *Shoot-Completing-Phase*, the Process11Shoot-call would attach an AD-1-query to each $x_{1,i}^\circ$ and an AD-3-query to each $y_{3,i}^\circ$. These constitute all the newly created AD-1- and AD-3-queries, which are indeed adjacent to $EB(x_1^\circ)$ – and thus adjacent to $EB(x_1)$. Moreover, it can be seen all these new AD-1- and AD-3-queries are adjacent to the (k_1, k_2)-alternated path in B_2 mentioned before, thus adjacent to $B_2(y_1)$. These establish claim (a).

Then, note that the newly created 1- and 3-queries adjacent to $EB(x_1^{\circ\circ})$ would not trigger calls to CollectTP (Indeed, these queries can only be heading towards B_2, and cannot form shoots nor MidTPs due to Inv2 and Inv3); all the CollectTP-calls are of the form $(1, x_{1,i}^\circ, y_{1,i}^\circ, -)$ and $(3, x_{3,i}^\circ, y_{3,i}^\circ, +)$ which meet $x_{1,i}^\circ, y_{3,i}^\circ \in EB(x_1)$ as well as $y_{1,i}^\circ, x_{3,i}^\circ \in B_2(y_1)$. These establish claim (b).

As mentioned, (ii) can then be proved via induction.

Finally, consider (iii). We already mentioned right after RandAssign$(1, y_1, -)$ returns x_1 it holds $\forall K : x_1 \notin ETable[K]$ by Inv5. This means $EB(x_1)$ contains no edges at this point. Thus all the edges in $EB(x_1)$ are added during the sub-calls. By Lemma 3, each sub-call to Process21TP, Process23TP, Process11Shoot, and Process33Shoot would "kill" all the E-queries newly created by it. We also note that these constitute all the sub-calls that are able to add E-queries into $EB(x_1)$. Thus the claim. □

The last lemma in this subsection states that the E-queries lying between certain 1-/3-queries must be dead.

Lemma 5. *At the end of each non-aborting simulator cycle, if two 1- or 3-queries not heading towards B_2 are adjacent to the same E-chain, then all the E-queries in this chain are dead.*

Proof. Consider the case of two 1-queries $(1, x_1^1, y_1^1, d, n)$ and $(1, x_1^{t+1}, y_1^{t+1}, d', n')$ first (by the assumption, $d, d' \neq \rightarrow$), and assuming an E-chain $(K_1, x_1^1, y_3^2, d_1, n_1), (K_2, x_1^3, y_3^2, d_2, n_2), \ldots, (K_t, x_1^{t+1}, y_3^t, d_t, n_t)$. Note that $d = \leftarrow \wedge d' = \leftarrow$ is not possible: if $d = \leftarrow$ then $n_1 > n$ and $d_1 = \rightarrow$ by Inv5, and further $n_2 > n_1$ and $d_2 = \leftarrow$,... This sequence finally yields $d_t = \leftarrow$, $n' > n_t > \ldots > n$, and thus $d' \neq \leftarrow$ by Inv5.

We then show that the claim holds for adapted queries. For this, wlog assume $n > n'$. As argued, this implies $d \neq \leftarrow$. Thus $d = \perp$. Assume that $(1, x_1^1, y_1^1, \perp, n)$ is created in a (long) simulator cycle triggered by D querying $\mathrm{P}i^\delta(z) \rightarrow z'$ with $num = n^*$; note that it necessarily be $n > n^*$.

We now argue that $n^* > n'$ is not possible. Otherwise, when G_2 receives the query $\mathrm{P}i^\delta$, $EB(x_1^{t+1})$ has already been table-defined. Thus: (i) by Lemma 4 (i), $z' \in EB(x_1^{t+1})$ is never possible; (ii) by Lemma 4 (ii), it must be $x_1^1 \in EB(z')$. By these, $x_1^1 \notin EB(x_1^{t+1})$, and the two assumed 1-queries can never be adjacent to the same E-chain.

By all the above, $(1, x_1^1, y_1^1, \perp, n)$ and $(1, x_1^{t+1}, y_1^{t+1}, d', n')$ could be adjacent to the same E-chain only if $n^* \leq n'$. We get two possibilities:

- If $n^* = n'$, then the query $(1, x_1^{t+1}, y_1^{t+1}, \leftarrow, n')$ is exactly the one that triggers the simulator cycle in question. By Lemma 4 (ii), it must be $x_1^1 \in EB(x_1^{t+1})$, and the E-edges between x_1^1 and x_1^{t+1} all lie in $EB(x_1^{t+1})$. Thus these E-queries are dead by Lemma 4 (iii) and our assumption on G_2's non-aborting;
- if $n^* < n'$, then both $(1, x_1^1, y_1^1, \perp, n)$ and $(1, x_1^{t+1}, y_1^{t+1}, d', n')$ are created during this cycle (due to $\mathrm{P}i^\delta(z) \rightarrow z'$). As $x_1^1 \in EB(z')$, it also holds $x_1^{t+1} \in EB(z')$. Thus similarly to the previous case, the E-queries between x_1^1 and x_1^{t+1} are dead after the cycle.

These conclude the case of two 1-queries. For all the other cases there's indeed no essential difference. □

7.6 B_2 is Acyclic

For readers familiar with the proof in [ABD+13a], it is easy to see the claim holds for the connected components formed by RA-2-edges. The difficulties lie in the AD-2-edges. However, AD-2-edges cannot be involved in MidTPs.

Proposition 4. *For any MidTP that is to be processed, the associated 2-query was necessarily created by* RandAssign*. This also means it was created due to D querying* P2 *or* $\mathrm{P2}^{-1}$.

Proof. We argue that it can never be an AD-2-query. For this, for an arbitrary AD-2-query $(2, x_2, y_2, \perp)$, assume that it was created when G_2 is completing the following path:

$$(K, k), (K, x_1, y_3), (1, x_1, y_1), (2, x_2, y_2), (3, x_3, y_3).$$

Then right before $(2, x_2, y_2)$ is in *Queries*, it already holds $x_2 \oplus k \in P_1^{-1}$ and $y_2 \oplus k \in P_3$. After $(2, x_2, y_2)$ is created, a call to UPDATECOMPLETED is made, which (if does not abort) adds $(1, K, x_1), (2, K, x_2), (3, K, x_3)$ to *Completed*.

If UPDATECOMPLETED aborts, then no further actions would happen after the creation of $(2, x_2, y_2)$. Otherwise, for any $k' : x_2 \oplus k' \in P_1^{-1}$, it falls into either of the following two cases:

- $k' = k$: then $(1, x_1, K)$ would not be processed again as $(1, K, x_1) \in \mathit{Completed}$;
- $k' \neq k$: then G_2 would take the queries as a 11-shoot to process rather than a 21-TP.

Thus no new 21-TP would be found around $(2, x_2, y_2)$. The argument for 23-TPs is similar by symmetry. Finally, according to the pseudocode, in chain-reaction calls, the 2-queries created by G_2 can only be adapted ones. Thus the involved $(2, x_2, y_2)$ was necessarily created due to D querying P2 or P2^{-1}. ◻

Then, note that by our code, each AD-2-edge is associated with a "mirror" E-query. More clearly, each time G_2 is to create an AD-2-query, it is completing a path, and the meta-data of the E-query corresponding to this path is kept as the meta-data of the AD-2-edges formed by this AD-2-query (e.g. the code of P1IN). Due to this assignment, we are able to prove an invariant for the edges in B_2.

Lemma 6. *At any point in any G_2 execution, for $en > en'$, there does not exist two edges $(y_1, x_3, k, \rightarrow, en)$ and $(y_1', x_3', k', ed', en')$ in B_2 such that $x_3 \oplus x_3' \in 4\mathcal{Z}$; there does not exist two edges $(y_1, x_3, k, \leftarrow, en)$ and $(y_1', x_3', k', ed', en')$ in B_2 such that $y_1 \oplus y_1' \in 4\mathcal{Z}$.*

The full power of this lemma will be used in Propositions 24 and 25. Moreover, since $\exists y_2, y_2' \in P_2^{-1} : x_3 \oplus k = y_2$ and $x_3' \oplus k' = y_2'$, this "invariant" is somewhat similar to Inv3. However, we correctly assign the meta-data ed, en, ed', en' to the two 2-edges – otherwise there's no means to state this "invariant".

Proof. Wlog we show there does not exist two edges $(y_1, x_3, k, \rightarrow, en)$ and $(y_1', x_3', k', ed', en')$ in B_2 such that $x_3 \oplus x_3' \in 4\mathcal{Z}$. To this end, let $(2, x_2, y_2, d_2, n_2)$ and $(2, x_2', y_2', d_2', n_2')$ be the 2-queries such that $x_2 = y_1 \oplus k$, $y_2 = x_3 \oplus k$, $x_2' = y_1' \oplus k'$, and $y_2' = x_3' \oplus k'$. We distinguish four cases.

Case 1: $d_2, d_2' \neq \perp$. Then $n_2 = en$, $n_2' = en'$, and $d_2 = ed = \rightarrow$, and the impossibility directly follows from Inv3.

Case 2: $d_2 = \perp$ while $d_2' \neq \perp$. Then $n_2' = en'$. Let $(K, x_1, y_3, \rightarrow, en)$ be the mirror E-query of $(2, x_2, y_2, \perp, n_2)$, and let $(1, x_1, y_1, d_1, n_1)$ and $(3, x_3, y_3, d_3, n_3)$ be the involved 1- and 3-queries. Then $n_3 > en$ and $d_3 \neq \rightarrow$ by Inv5, thus $n_3 > en > en'/n_2'$.

Next, a crucial point is that *the 1-/3-query lies between the heads of an AD-2-edge and its mirror E-query must head towards B_2*. More clearly, we argue that it cannot be $d_3 = \perp$ (so that $d_3 = \leftarrow$), by eliminating both of the two possibilities of the pair $((K_1, x_1, y_3, \rightarrow), (3, x_3, y_3, \perp))$:

- $(3, x_3, y_3, \perp)$ is created in a call to PROCESS21TP(x_1, y_1, K). Then by Proposition 4, the 2-query $(2, x_2, y_2)$ cannot be an adapted one;
- $(3, x_3, y_3, \perp)$ is created in a call to PROCESSSHOOT. Then by Lemma 4 (ii), the E-query (K, x_1, y_3) necessarily belongs to the new E-chain of this call, and thus cannot be the mirror E-query of any AD-2-query.

Thus $d_3 \neq \perp$; thus $d_3 = \leftarrow$ as argued. Then $x_3 \oplus x_3' \in 4\mathcal{Z}$ is not possible, as otherwise we got $x_3 \oplus y_2' \in 5\mathcal{Z}$ and contradict Inv2.

Case 3: $d_2 \neq \perp$ while $d_2' = \perp$. Then $n_2 = en$. Let $(K', x_1', y_3', ed', en')$ be the mirror E-query of $(2, x_2', y_2', d_2', n_2')$, and let $(1, x_1', y_1', d_1', n_1')$ and $(3, x_3', y_3', d_3', n_3')$ be the involved 1- and 3-queries. We exclude two possibilities:

- If $en/n_2 > n_3'$, then $y_2 \oplus x_3' = x_3 \oplus k \oplus x_3' \in 5\mathcal{Z}$ contradicts Inv2;
- If $n_3' > en/n_2$, then $n_3' > en/n_2 > en'$. By the pseudocode, we know that the creation of $(2, x_2, y_2, \rightarrow, n_2)$ must be an "isolated" simulator cycle. (The case has to be: D makes a query to P2, G_2 does not detect any tripwire, and calls RANDASSIGN. In this cycle, no chain would be completed, and only one (2-)query is created.) Thus $(3, x_3', y_3', d_3', n_3')$ is created in a later cycle, and thus $d_3' \neq \perp$ (because each later-created AD-3-query is adjacent to some connected component $EB(z)$ which satisfies $y_3' \notin EB(z)$ as y_3' has been table-defined). Also $d_3' \neq \rightarrow$ by Inv5, thus $d_3' = \leftarrow$, and the impossibility finally follows from Inv2.

Case 4: $d_2 = d_2' = \perp$. Let $(K, x_1, y_3, \to, en)$ be the mirror E-query of $(2, x_2, y_2, d_2, n_2)$, and let $(1, x_1, y_1, d_1, n_1)$ and $(3, x_3, y_3, d_3, n_3)$ be the involved 1- and 3-queries; let $(K', x_1', y_3', ed', en')$ be the mirror of $(2, x_2', y_2', d_2', n_2')$, and let $(1, x_1', y_1', d_1', n_1')$, $(3, x_3', y_3', d_3', n_3')$ be the involved 1- and 3-queries. Then $d_3 = \leftarrow$ as argued in *Case 2*, and $n_3 > en > en'$. We also exclude two possibilities as follows.

First, if $en > n_3'$, then $n_3 > en > n_3'$, and the impossibility follows from Inv3;

Second, if $n_3' > en$, then $n_3' > en > en'$, and $d_3' \neq \to$ by Inv5. Note that if $d_3' = \leftarrow$ then the impossibility directly follows from Inv3. Thus we proceed to argue $d_3' \neq \perp$. For this consider two possibilities:

– If $(K', x_1', y_3', ed', en')$ and $(3, x_3', y_3', d_3', n_3')$ are not created in the same cycle, then as argued in *Case 3*, $d_3' \neq \perp$;
– If $(K', x_1', y_3', ed', en')$ and $(3, x_3', y_3', d_3', n_3')$ are indeed created in the same cycle, then they must be created in the same chain-reaction call: because after the chain-reaction call during which (K', x_1', y_3') is created, unless abortion occurs, (K', x_1', y_3') should have been dead by Lemma 3, which implies $(3, x_3', y_3') \in Queries$. By this, $d_3' = \perp$ is already excluded: by the pseudocode, the only possibility for G_2 first creating an E-query and then creating an AD-3-query adjacent to this E-query is in a call to ProcessShoot;[10] but in this case, the E-query lies in the new E-chain, and thus (K', x_1', y_3') cannot have been the mirror E-query of $(2, x_2', y_2', d_2', n_2')$.

The above complete the proof. ◻

Finally we are able to prove that B_2 does not contain any cycles either.

Lemma 7. *Connected components of B_2 are directed trees with edges directed away from the root, and the num values on the edges of any directed path in B_2 are strictly increasing.*

Proof. The proof follows the same line as Proposition 1, with the help of Lemma 6. ◻

At any point, given a node x_1 (or y_3) in EB, we denote by $Tr(x_1)$ ($Tr(y_3)$, resp.) the (time-dependent) tree obtained by "dangling" the connected component $EB(x_1)$ ($EB(y_3)$, resp.) by x_1 (y_3, resp.), such that x_1 (y_3, resp.) is the root. Similarly, given a node y_1 (or x_3) in B_2, we write $Tr(y_1)$ ($Tr(x_3)$, resp.) for the (time-dependent) tree obtained by "dangling" the connected component $B_2(y_1)$ ($B_2(x_3)$, resp.) by y_1 (x_3, resp.).

We would frequently refer the subtrees of some certain tree (either in EB or in B_2). For this, for a tree T and a node z in T, we write $SubT(T, z)$ for the *subtree* of T rooted at z; if z is the root, then $SubT(T, z) = T$.

We have another corollary: the same 2-query cannot be involved in two distinct MidTPs.

Proposition 5. *The same 2-query $(2, x_2, y_2)$ cannot be involved in two distinct detected MidTPs.*

Proof. If the two MidTPs are not detected in the same cycle, then after G_2 processing the earlier-detected MidTP, $(2, x_2, y_2)$ must be in a complete path (since non-aborting), and following the same line as Proposition 4 we know it cannot be involved in MidTPs any more. Thus the two MidTPs are detected in the same cycle. By the code, the only cycle that can meet this requirement is the long cycle. Assume that this cycle is induced by D querying $\mathrm{P}i^{\delta}(z) \to z'$ ($(i, \delta) \in \{(1, -), (3, +)\}$). We exclude two possibilities.

Case 1: the two MidTPs are two 21- or 23-TPs. Wlog consider the case of G_2 detecting two 21-TPs induced by creating $(1, x_1, y_1, d_1)$ and $(1, x_1', y_1', d_1')$. This means $\exists k \neq k' \in \mathcal{Z} : y_1 \oplus k = x_2$ and $y_1' \oplus k' = x_2$. This implies $y_1 \oplus y_1' = k \oplus k'$. In long cycles, no 2-query with $dir \neq \perp$ can be created, thus by Proposition 4, $(2, x_2, y_2)$ existed before this cycle, and by Inv2 we have $d_1, d_1' \neq \to$. By Lemma 4 (ii) we got $y_1, y_1' \in B_2(z)$ – note that it might be $y_1 = z$ or $y_1' = z$, however this does not hinder the claim. Thus right before G_2 detecting the later MidTP, there exists a "pseudo-cycle" in B_2: $y_1 - \ldots - y_1' \xrightarrow{\oplus k \oplus k'} (y_1)$. We exclude two possibilities:

(i) The path between y_1 and y_1' is directed from y_1 to y_1': $y_1 \to x_3^* \to \ldots \to x_3^{**} \to y_1'$ (for some x_3^*, x_3^{**}). Then by Lemma 7, the edge between x_3^{**} and y_1' necessarily has *enum* larger than that of the edge between x_3^* and y_1, and thus $y_1'' = y_1 \oplus k \oplus k'$ is not possible by Lemma 6.
When the path is directed from y_1' to y_1, the argument is indeed similar.
(ii) There exists a vertex z^* such that the path is directed from z^* to y_1, and from z^* to y_1': $y_1 \leftarrow x_3^* \leftarrow \ldots \leftarrow z^* \to \ldots \to x_3^{**} \to y_1'$ (for some x_3^*, x_3^{**}). Then $y_1' = y_1 \oplus k \oplus k'$ is not possible by Lemma 6.

The above contradiction with Lemma 6 indeed indicates that G_2 necessarily aborted before creating the later 1-query and detecting the later MidTP, and this contradicts our (implicit) non-aborting assumption.

[10] It cannot have been a call to Process21TP, because otherwise $(2, x_2', y_2')$ cannot have been an adapted one due to Proposition 4.

Case 2: the two MidTPs are a 21-TP and a 23-TP. Assume that the 21-TP is induced by G_2 creating $(1, x_1, y_1, d_1)$ with $y_1 = x_2 \oplus k$, while the 23-TP is induced by G_2 creating $(3, x_3, y_3, d_3)$ with $x_3 = y_2 \oplus k'$. Similarly to *Case 1*, $d_1 \neq\rightarrow$ and $d_3 \neq\leftarrow$, and $y_1 \in B_2(z)$ and $x_3 \in B_2(z)$. Let $x'_3 = y_2 \oplus k$, then $x'_3 = x_3 \oplus k \oplus k'$, and right before G_2 detecting the later MidTP, there exists a "pseudo-cycle" in B_2: $x'_3 - y_1 - \ldots - x_3 \overset{\oplus k \oplus k'}{\overline{\quad\quad}} (x'_3)$. Thus the impossibility is reached similarly to *Case 1*. □

Finally, MidTPs and Shoots are somewhat "mutual exclusive".

Proposition 6. *During a long simulator cycle, assume that when G_2 is processing a MidTP, it completes a chain corresponding to $(K, x_1^\circ, y_3^\circ)$ without abortion. Then G_2 would not process any shoot of the form $(1, x_1^\circ, \{K, K'\})$ or $(3, y_3^\circ, \{K, K'\})$ $(K' \neq K)$ in this cycle.*

Proof. Wlog consider the case of processing a 21-TP $(1, K, x_1^\circ)$, and assume the involved path is

$$(K, k), (K, x_1^\circ, y_3^\circ), (1, x_1^\circ, y_1^\circ), (2, x_2^\circ, y_2^\circ, d_2^\circ, n_2^\circ), (3, x_3^\circ, y_3^\circ, \perp, n_3^\circ).\ (y_1^\circ \oplus x_2^\circ = y_2^\circ \oplus x_3^\circ = k)$$

By Proposition 4 we have $n_2^\circ < \mathit{cycleStartNum}$ (recall that $\mathit{cycleStartNum}$ is the qnum value of the query which sets off this long cycle). Then G_2 clearly would not detect any 11-shoots of the form $(1, x_1^\circ, \{K, K'\})$ after it creates $(1, x_1^\circ, y_1^\circ)$, as otherwise it holds $y_1^\circ \oplus k \oplus k' \in P_1^{-1}$ and G_2 should have not detected $(1, K, x_1^\circ)$.

On the other hand, assume that G_2 detects a 33-shoot $(3, y_3^\circ, \{K, K'\})$ after creating $(3, x_3^\circ, y_3^\circ)$. This indicates the existence of a 3-query $(3, x'_3, y'_3, d'_3, n'_3)$ with $x'_3 = x_3^\circ \oplus k \oplus k' = y_2^\circ \oplus k'$. It necessarily be $n'_3 < \mathit{cycleStartNum}$, as otherwise G_2 detecting a new 33-shoot formed by $(3, x_3^\circ, y_3^\circ)$ and $(3, x'_3, y'_3)$ would lead to abortion in CollectTP, and thus G_2 would not "process" the 33-shoot. Thus $(3, x'_3, y'_3)$ along with $(2, x_2^\circ, y_2^\circ)$ indicate $x_2^\circ \oplus k' = y_1^\circ \oplus k \oplus k' \in P_1^{-1}$ by Inv6, and after creating $(1, x_1^\circ, y_1^\circ)$, G_2 should have detected $(1, x_1^\circ, \{K, K'\})$ rather than $(1, K, x_1^\circ)$, a contradiction. These establish the claim for 21-TPs. □

7.7 Properties Around *DUShoots*

First, we reconsider the conditions for ProcessShoot to add new tuples to *DUShoots*. For conceptual convenience, we imagine the call "extends" the old and the new E-chains simultaneously. Then we note that for some pair of values $(x'_{1,i}, x_{1,i})$ (in the old and new E-chains, resp.), if it holds $x'_{1,i} \notin \mathit{ETable}[K]$ for the corresponding K, then G_2 would add the 33-shoot "anchored" at the "next" pair $(y'_{3,i}, y_{3,i})$ to *DUShoots*. The intuition is the value $y'_{3,i} \leftarrow \mathbf{E}.\mathrm{E}(K, x'_{1,i})$ is indeed fresh in this case. However, recalling from Section 6 that *for a shoot in DUShoots, we wish both of the two involved queries are fresh.* Thus there seems a contradiction.

However, our design is sound: the rationale is that for a pair of corresponding values from the old and the new E-chain, *if the value from the old one is not in ETable, then the value from the new one is not in ETable either.*

Lemma 8. *Consider the Make-E-Chain-Phase of a call to* Process11Shoot(x_1, y_1, K_1, K_2). *Following the notations in the pseudocode, in the first iteration, for each i, if $x'_{1,i+1} \notin \mathit{ETable}[K_1]$, then the corresponding value in the new E-chain $x_{1,i+1}$ would not be in $\mathit{ETable}[K_2]$ either; if $y'_{3,i} \notin \mathit{ETable}[K_2]^{-1}$, then the corresponding $y_{3,i}$ would not be in $\mathit{ETable}[K_1]^{-1}$. In the second iteration, for each i, if $x'_{1,i} \notin \mathit{ETable}[K_2]$ ($y'_{3,i} \notin \mathit{ETable}[K_1]^{-1}$, resp.), then the corresponding $x_{1,i}$ ($y_{3,i}$, resp.) would not be in $\mathit{ETable}[K_1]$ ($\mathit{ETable}[K_2]^{-1}$) either. Similar claim holds for* Process33Shoot*-calls.*

Proof. Wlog consider a pair $(x'_{1,i+1}, x_{1,i+1})$ in the first iteration of Process11Shoot(x_1, y_1, K_1, K_2). To show the claim, we argue once $x_{1,i+1} \in \mathit{ETable}[K_2]$ then it must hold $x'_{1,i+1} \in \mathit{ETable}[K_1]$. We distinguish two cases: the Process11Shoot-call is triggered by D directly querying $\mathrm{P1}^{-1}$, or by an AD-1-query.

Case 1: Process11Shoot(x_1, y_1, K_1, K_2) *happens in a cycle due to D querying* $\mathrm{P1}^{-1}(y_1)$, *and* $x_{1,i+1} = x_1$. Then right after RandAssign in $\mathrm{P1}^{-1}(y_1)$ return x_1, it holds $\forall K, x_1 \notin \mathit{ETable}[K]$. By the code, all the chain-reaction calls made before Process11Shoot(x_1, y_1, K_1, K_2) are of the form Process11Shoot(x_1, y_1, K_3, K_4). Thus if G_2 finds $x_1 \in \mathit{ETable}[K_2]$ in Process11Shoot(x_1, y_1, K_1, K_2), there necessarily be an earlier call to Process11Shoot(x_1, y_1, K_2, K_3) with $K_3 \neq K_1$. These imply the existence of two 1-queries $(1, x'_1, y'_1)$ and $(1, x''_1, y''_1)$ with $y'_1 = y_1 \oplus k_1 \oplus k_2$ and $y''_1 = y_1 \oplus k_2 \oplus k_3$. Thus $y'_1 \oplus y''_1 = k_1 \oplus k_3$. The two 1-queries necessarily existed before this cycle and $x'_1, x''_1 \notin \mathit{Border}$, as otherwise G_2 would have aborted in CollectTP when detecting 11-shoots formed by $(1, x_1, y_1)$ and them. Thus $(1, K_1, x'_1) \in \mathit{Completed}$ by Inv8 and Inv7, and $x'_1 \in \mathit{ETable}[K_1]$ by Lemma 1. As we assumed $x_{1,i+1} = x_1$, we got $x'_{1,i+1} = x'_1$; thus the claim.

Case 2: $\textsc{Process11Shoot}(x_1, y_1, K_1, K_2)$ *is in a cycle due to* D *querying* $\mathrm{P}i^{\delta}(z) \to z'$ $((i,\delta) \in \{(1,-),(3,+)\})$, *with* $z \neq x_{1,i+1}$. Then by Lemma 4 (i), it holds $x_{1,i+1} \in EB(z')$, and the path between z' and $x_{1,i+1}$ is directed from z' to $x_{1,i+1}$. Thus there exists an E-query of the form $(K^*, x_{1,i+1}, y_3^*, \leftarrow)$, and right after $x_{1,i+1} \in EB(z')$ holds, it holds $\forall K \neq K^*, x_{1,i+1} \notin ETable[K]$. By Proposition 6, the E-query $(K_2, x_{1,i+1}, y_{3,i})$ cannot be created during G_2 processing a MidTP. Thus by Lemma 4 (ii), G_2 necessarily popped (and processed) a shoot equivalent to $(1, x_{1,i+1}, \{K_2, K_3\})$ with $K_3 \neq K_1$. These imply the existence of two 1-queries $(1, x'_{1,i+1}, y'_{1,i+1})$ and $(1, x''_{1,i+1}, y''_{1,i+1})$ with $y'_{1,i+1} = y_{1,i+1} \oplus k_1 \oplus k_2$ and $y''_{1,i+1} = y_{1,i+1} \oplus k_2 \oplus k_3$. Thus $y'_{1,i+1} \oplus y''_{1,i+1} = k_1 \oplus k_3$. The two 1-queries necessarily existed before this cycle and $x'_{1,i+1}, x''_{1,i+1} \notin Border$, as otherwise G_2 would have aborted in $\textsc{CollectTP}$ when detecting 11-shoots formed by $(1, x_{1,i+1}, y_{1,i+1})$ and them. Thus similarly to *Case 1*, the claim holds. ◻

By these, shoots in $DUShoots$ have regular structures.

Proposition 7. *At any point in a* G_2 *execution, for any tuple* $(1, \{(x_1, y_1), (x'_1, y'_1)\}) \in DUShoots$, *it holds:*

(i) $\exists K, K', y_3,$ *and* $y'_3 : (K, x_1, y_3, \leftarrow), (K', x'_1, y'_3, \leftarrow) \in EQueries$, *and* $y_1 \oplus y'_1 = k \oplus k'$;
(ii) $(1, x_1, y_1, d), (1, x'_1, y'_1, d') \in Queries$, *and one of* d *and* d' *equals* $\to$, *while the other equals* $\perp$;
(iii) For any $k'' \notin \{k, k'\}$, $y_1 \oplus k \oplus k'' \notin P_1^{-1}$.

Symmetrically, for any tuple $(3, \{(x_3, y_3), (x'_3, y'_3)\}) \in DUShoots$, *it holds:*

(i) $\exists K, K', x_1,$ *and* $x'_1 : (K, x_1, y_3, \to), (K', x'_1, y'_3, \to) \in EQueries$, *and* $x_3 \oplus x'_3 = k \oplus k'$;
(ii) $(3, x_3, y_3, d), (3, x'_3, y'_3, d') \in Queries$, *and one of* d *and* d' *equals* $\leftarrow$, *while the other equals* $\perp$;
(iii) For any $k'' \notin \{k, k'\}$, $x_3 \oplus k \oplus k'' \notin P_3$.

Proof. Wlog consider a tuple $(1, \{(x_1, y_1), (x'_1, y'_1)\}) \in DUShoots$. From the code we know such a tuple can only be added to $DUShoots$ in $\textsc{ProcessShoot}$. Wlog consider a call to $\textsc{Process11Shoot}(x_1^{\circ}, y_1^{\circ}, K_1, K_2)$, let $x_1^{\circ\circ} = P_1^{-1}(y_1^{\circ} \oplus k_1 \oplus k_2)$, and assume that $x_1 \in EB(x_1^{\circ\circ})$. Then by the conditions around the set $NewDUShootSet$, it can be seen that $x_1 \in EB(x_1^{\circ\circ})$ does not hold before $\textsc{Process11Shoot}(x_1^{\circ}, y_1^{\circ}, K_1, K_2)$ is made. Thus the query that brings x_1 into $EB(x_1^{\circ\circ})$ is either of the form $(K_1, x_1, y_3, \leftarrow)$ or $(K_2, x_1, y_3, \leftarrow)$ for some y_3. Wlog assume this query is $(K_1, x_1, y_3, \leftarrow)$. Then by Lemma 8 it implies the existence of $(K_2, x'_1, y'_3, \leftarrow)$ for some y'_3. These establish (i).

Based on (i), right after $x_1 \in EB(x_1^{\circ\circ})$ holds, it holds $x_1 \notin P_1$ by Inv5. Thus by the code, G_2 soon creates a 1-query $(1, x_1, y_1, \to)$. At this point, it holds $\forall z \in 2\mathcal{Z}\backslash\{0\}, y_1 \oplus z \notin P_1^{-1}$ by Inv3. G_2 then creates the AD-1-query $(1, x'_1, y'_1, \perp)$ with $y'_1 = y_1 \oplus k_1 \oplus k_2$ (if abortion does not occur). These establish (ii), and show that (iii) holds right after a tuple is added to $DUShoots$.

We then proceed to argue that (iii) keeps holding after a tuple is added to $DUShoots$. For this, we consider each case of G_2 creating a new 1-query $(1, x''_1, y''_1)$ with $y''_1 = y_1 \oplus k \oplus k''$ for $k'' \neq k, k'$. In some of the cases (e.g. *Case 1* below), it's not possible to form such a structure; in the others (e.g. *Case 2*), the tuple has been removed from $DUShoots$.

Case 1: $(1, x''_1, y''_1)$ *is created as the result of* D *querying* $\mathrm{P}1$, *or a short simulator cycle (cf. subsection 7.1).* However, 1-queries created in these cases are necessarily with $dir = \to$, and cannot have $y''_1 \oplus y_1 \in 2\mathcal{Z}$ by Inv3;

Case 2: $(1, x''_1, y''_1)$ *is created as the result of* $\textsc{RandAssign}(1, y''_1, -)$ *(after* D *querying* $\mathrm{P}1^{-1}(y''_1)$*).* In this case, $(1, \{(x_1, y_1), (x'_1, y'_1)\})$ must have been removed from $DUShoots$, otherwise $y''_1 \oplus y_1 \in 2\mathcal{Z}$ would have caused G_2 abort in the call $\textsc{CheckDUnaware}(y''_1, Y1)$;

Case 3: $(1, x''_1, y''_1)$ *is an AD-1-query created as the result of* G_2 *processing a 23-TP.* By the code, G_2 would call $\textsc{CheckDUnaware}(y''_1, Y1)$ before trying to create it, and would abort since $y''_1 = y_1 \oplus k \oplus k''$, thus would not create it.

Case 4: $(1, x''_1, y''_1)$ *is an AD-1-query created in a later* $\textsc{ProcessShoot}$*-call.* Assume that in this call, the 1-query that forms a shoot with $(1, x''_1, y''_1)$ is $(1, x'''_1, y'''_1)$. Thus $(1, x'''_1, y'''_1)$ cannot be newly created in this later $\textsc{ProcessShoot}$-call, as otherwise $y'''_1 \oplus y_1 \in 4\mathcal{Z}$ can be inferred from $y''_1 \oplus y_1 \in 2\mathcal{Z}$, contradicting Inv3. Then it necessarily falls into two cases:

(i) $(1, x'''_1, y'''_1) = (1, x_1, y_1)$ or $(1, x'_1, y'_1)$. This implies in this later $\textsc{ProcessShoot}$-call, G_2 obtains x_1 or x'_1 when evaluating along the old E-chain; by the pseudocode of $\textsc{ProcessShoot}$, this necessarily cause G_2 remove $(1, \{(x_1, y_1), (x'_1, y'_1)\})$ from $DUShoots$ right before creating $(1, x''_1, y''_1)$;

(ii) $(1, x_1''', y_1''') \neq (1, x_1, y_1), (1, x_1', y_1')$. Then by the code, as $(1, x_1''', y_1''')$ exists before the later PROCESSSHOOT-call, G_2 would deem it as "D-aware", and would call CHECKDUNAWARE$(y_1'', Y1)$ before trying to create $(1, x_1'', y_1'')$, and would abort since $y_1'' = y_1 \oplus k \oplus k''$.

By the above, (iii) keeps holding, unless $(1, \{(x_1, y_1), (x_1', y_1')\})$ is removed from *DUShoots*. These complete the proof. □

Proposition 7 (iii) proves the soundness of our implementation of REMOVEDUSHOOTS. For this, note that if there exist two tuples $(1, \{(x_1, y_1), (x_1', y_1')\})$ and $(1, \{(x_1, y_1), (x_1'', y_1'')\})$ in *DUShoots* (i.e. they "share" the 1-query $(1, x_1, y_1)$), then the current REMOVEDUSHOOTS procedure may fail to identify all shoots that should be removed from *DUShoots*.

Queries in *DUShoots* cannot form interesting shoots.

Proposition 8. *Right before any call to* PROCESS11SHOOT(x_1, y_1, K_1, K_2), *let* $x_1' = P_1^{-1}(y_1 \oplus k_1 \oplus k_2)$, *then both* DAWARENESS$(x_1, X1)$ *and* DAWARENESS$(x_1', X1)$ *equal 1; symmetrically, right before any* PROCESS33SHOOT(x_3, y_3, K_1, K_2), DAWARENESS *returns 1 on both* y_3 *and* $y_3' = P_3(x_3 \oplus k_1 \oplus k_2)$.

Proof. Wlog consider such a call to PROCESS11SHOOT(x_1, y_1, K_1, K_2). This call is necessarily due to G_2 popping a shoot $(1, x_1, \{K_1, K_2\})$ from *ShootQueue* such that $(1, x_1, \{K_1, K_2\}) \notin$ *ProcessedShoot*. By the pseudocode, it's necessarily due to G_2 creating a 1-query $(1, x_1, y_1)$ and then detecting $(1, x_1', y_1', d_1', n_1')$ s.t. $y_1 \oplus y_1' = k_1 \oplus k_2$. Under these assumptions, we consider each case where G_2 would create $(1, x_1, y_1)$:

Case 1: D directly queries $\mathrm{P1}^{-1}(y_1)$. In this case DAWARENESS$(x_1, X1)$ clearly equals 1 before the PROCESS11SHOOT-call. On the other hand, if DAWARENESS$(x_1', X1) = 0$, then since $y_1 \oplus y_1' = k_1 \oplus k_2 \in 2\mathcal{Z}$, D querying $\mathrm{P1}^{-1}(y_1)$ would have caused G_2 abort in CHECKDUNAWARE$(y_1, Y1)$. Thus DAWARENESS$(x_1', X1) = 1$ before the call.

Case 2: G_2 *creates* $(1, x_1, y_1, \perp)$ *in a call to* PROCESS23TP(x_3, y_3, K). Then since G_2 would not add any shoots containing $(1, x_1, y_1)$ to *DUShoots* after creating $(1, x_1, y_1)$, it holds DAWARENESS$(x_1, X1) = 1$ before the PROCESS11SHOOT-call. On the other hand, if DAWARENESS$(x_1', X1) = 0$, then the fact that $y_1 \oplus y_1' = k_1 \oplus k_2$ would have caused G_2 abort in CHECKDUNAWARE$(y_1, Y1)$ before trying to create $(1, x_1, y_1)$ (note that by assumption, when $(1, x_1, y_1)$ is created, $(1, x_1', y_1') \in$ *Queries* already holds).

Case 3: G_2 *creates* $(1, x_1, y_1, \perp)$ *in a* PROCESSSHOOT-*call.* Wlog assume that this call is PROCESS11SHOOT(x_1^*, y_1^*, K_3, K_4), and the shoot leading to this call is $(1, x_1^*, \{K_3, K_4\})$.

If $\{K_3, K_4\} = \{K_1, K_2\}$, then it's not hard to see $(1, x_1^*, \{K_1, K_2\}) \equiv (1, x_1, \{K_1, K_2\})$ (discarding the notations K_3 and K_4). This implies $(1, x_1, \{K_1, K_2\})$ would be in *ProcessedShoot* after PROCESS11SHOOT(x_1^*, y_1^*, K_1, K_2) returns, thus the purported call to PROCESS11SHOOT(x_1, y_1, K_1, K_2) would not have been possible. By this, it has to be $\{K_3, K_4\} \neq \{K_1, K_2\}$.

We then assume that in PROCESS11SHOOT(x_1^*, y_1^*, K_3, K_4), the 1-query corresponding to creating $(1, x_1, y_1, \perp)$ is $(1, x_1^\circ, y_1^\circ, d_1^\circ, n_1^\circ)$. Furthermore, assume that G_2 computes x_1° via $\mathrm{EIN}^{-1}(K_3, y_3^\circ)$. It necessarily be $y_3^\circ \in ETable[K_3]^{-1}$ before the call PROCESS11SHOOT(x_1^*, y_1^*, K_3, K_4), as otherwise y_1 would be somewhat random and could not form new interesting shoots.[11] However, by the code of PROCESS11SHOOT, since $y_3^\circ \in ETable[K_3]^{-1}$, $(1, \{(x_1, y_1), (x_1^\circ, y_1^\circ)\})$ would not be added to *DUShoots* and thus DAWARENESS$(x_1, X1) = 1$.

Then, similarly to *Case 2*, if DAWARENESS$(x_1', X1) = 0$, then $y_1 \oplus y_1' = k_1 \oplus k_2$ would have caused G_2 abort in CHECKDUNAWARE$(y_1, Y1)$ before trying to create $(1, x_1, y_1)$ (note that G_2 would call CHECKDUNAWARE$(y_1, Y1)$ because $y_3^\circ \in ETable[K_3]^{-1}$).

The above complete the proof. □

Proposition 9. *In any simulator cycle, a tuple* $(1, \{(x_1, y_1), (x_1', y_1')\})$ *or* $(3, \{(x_3, y_3), (x_3', y_3')\})$ *cannot first be added to DUShoots while then be removed.*

[11] If $y_3^\circ \notin ETable[K_3]^{-1}$ then PROCESS11SHOOT(x_1^*, y_1^*, K_3, K_4) would create a new E-query $(K_3, x_1^\circ, y_3^\circ, \leftarrow)$, right after which $x_1^\circ \notin P_1$ by Inv5, and thus PROCESS11SHOOT(x_1^*, y_1^*, K_3, K_4) would create a new 1-query $(1, x_1^\circ, y_1^\circ, \rightarrow)$. Thus by Inv3, after PROCESS11SHOOT(x_1^*, y_1^*, K_3, K_4) returns, $(1, x_1^\circ, y_1^\circ)$ is the only 1-query satisfying $y_1 \oplus y_1^\circ \in 2\mathcal{Z}$.

Proof. Wlog consider such a tuple $(1, \{(x_1, y_1), (x_1', y_1')\})$, and assume: (i) $y_1 \oplus y_1' = k \oplus k'$; (ii) $(1, x_1, y_1)$ is "anchored" at the old E-chain corresponding to the ProcessShoot-call which adds $(1, \{(x_1, y_1), (x_1', y_1')\})$ to *DUShoots*; (iii) in this ProcessShoot-call, G_2 creates two E-queries $(K, y_3, x_1, \rightarrow)$ and $(K', \overline{y_3}, x_1, \leftarrow)$, cf. Fig. 5 (left); (iv) this ProcessShoot-call happens in a long cycle due to D querying $\mathrm{P1}^{-1}(y_1^\circ) \rightarrow x_1^\circ$ (this is wlog). Then by Inv4 and the code, after this ProcessShoot-call returns, it holds $x_1 \notin ETable[K^*]$ for any $K^* \neq K, K'$.

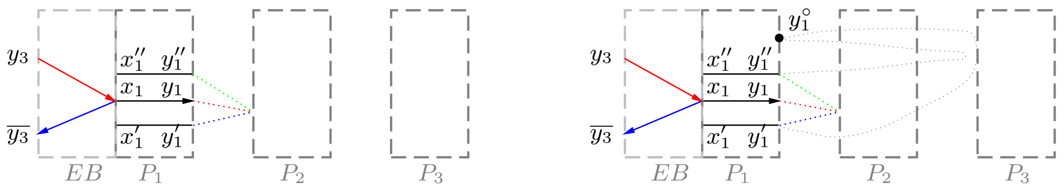


Fig. 5. For Proposition 9: lines in red, blue, and lime indicate edges with (K, k), (K', k'), and (K'', k'') respectively. (right) illustration of the "pseudo-cycle".

Now, if $(1, \{(x_1, y_1), (x_1', y_1')\})$ is later removed from *DUShoots*, then G_2 necessarily "reaches" $(1, x_1, y_1)$ via either (K, y_3, x_1) or $(K', \overline{y_3}, x_1)$ when it is evaluating along the old E-chain of a later ProcessShoot-call. Wlog assume that the later ProcessShoot-call shares the key K with the earlier ProcessShoot-call, and the other key of the later ProcessShoot-call is $K'' \neq K, K'$. Then in the earlier ProcessShoot-call, G_2 created an AD-1-query $(1, x_1', y_1', \perp)$ with $y_1' = y_1 \oplus k \oplus k'$, while in later ProcessShoot-call, G_2 is to create an AD-1-query $(1, x_1'', y_1'', \perp)$ with $y_1'' = y_1 \oplus k \oplus k''$. Thus $y_1' \oplus y_1'' = k' \oplus k'' \in 2\mathcal{Z}$. By Lemma 4 (ii), both y_1' and y_1'' are in $B_2(y_1^\circ)$. Therefore, right before G_2 is to remove $(1, \{(x_1, y_1), (x_1', y_1')\})$ (and then create $(1, x_1'', y_1'', \perp)$), a "pseudo-cycle" $y_1^\circ - \ldots - y_1' \xrightarrow{\oplus k' \oplus k''} y_1'' - \ldots - y_1^\circ$ exists in B_2, cf. Fig. 5 (right), which contradicts Lemma 6 (similarly to Proposition 5). This implies G_2 should have aborted at some earlier point, and would not remove $(1, \{(x_1, y_1), (x_1', y_1')\})$. Thus the claim. $\square$

Remark 1. Consider the previous proof. Assume that $(1, x_1'', y_1'', \perp)$ is created later than $(1, x_1', y_1', \perp, n_1')$. Then as $n_1' > cycleStartNum$, G_2 would abort in CollectTP$(1, x_1'', y_1'')$ after creating $(1, x_1'', y_1'')$. However, at this point, $(1, \{(x_1, y_1), (x_1', y_1')\})$ has been removed. This explains why we take the above more complicated pseudo-cycle-based proof – we'd like to show that G_2 would abort *before* removing $(1, \{(x_1, y_1), (x_1', y_1')\})$.

Consider an E-chain $z_1 - \ldots - z_i - \ldots - z_l$ (informally). If the DAwareness function values of both z_1 and z_l equal 1 while the DAwareness value of z_i equals 0 for some $1 < i < l$, then we call this chain *bad*. Such bad E-chains in fact never exist. The proof relies on three propositions.

Proposition 10. *When an E-chain is originally created, it cannot be a bad one.*

Proof. We note that if the DAwareness function values of some nodes in an E-chain are 0, then some parts of the E-chain were necessarily created in ProcessShoot. Wlog assume that there exists a value x_1 such that DAwareness$(x_1, X1) = 0$. Further assume that x_1 is in the old E-chain of this ProcessShoot-call (this assumption is wlog because the return values of DAwareness on both the new E-chain and the old E-chain are determined by the state of the old E-chain). Then by the code, it can be seen that there necessarily exists a query $(K, x_1, y_3, \leftarrow)$ such that when the ProcessShoot-call computes y_3 (in the *Make-E-Chain-Phase*), it finds $y_3 \notin ETable[K]^{-1}$. Thus right after (K, x_1, y_3) is created and x_1 is in this chain, it holds $\forall K' \neq K, x_1 \notin ETable[K']$ by Inv4. Thus DAwareness returns 0 on all the nodes in the tree $SubT(Tr(y_3), x_1)$ (cf. page 33), and x_1 cannot be the purported "turning point" z_i. $\square$

Proposition 11. *Consider G_2 creating a new E-query. It cannot be that an E-chain was good before this creating action, but turns bad after it.*

Proof. Towards a contradiction, wlog assume that there is a node x_1 in an E-chain such that

– DAwareness$(x_1, X1) = 0$, and
– $x_1 \notin ETable[K]$ for some K,

whereas later G_2 creates an E-query $(K, x_1, y_3, \rightarrow)$, after which DAwareness$(x_1, X1)$ remains 0 while DAwareness$(y_3, Y3) = 1$. To show the impossibility, we exclude each possibility of G_2 creating $(K, x_1, y_3, \rightarrow)$:

Case 1: D querying $\mathrm{E}(K, x_1)$. This is clearly not possible, as if $\mathrm{DAWARENESS}(x_1, X1) = 0$ then D querying $\mathrm{E}(K, x_1)$ would have caused G_2 abort in $\mathrm{CHECKDUNAWARE}(x_1, X1)$.

Case 2: D querying $\mathrm{P2}(x_2)$ *for some x_2 and $k \in \mathcal{Z}$ s.t. $x_1 = P_1^{-1}(k \oplus x_2)$.* Similarly, $\mathrm{DAWARENESS}(x_1, X1) = 0$ would have caused G_2 abort in $\mathrm{CHECKDUNAWARE}(x_2, X2)$.

Case 3: A call to $\mathrm{PROCESS21TP}(x_1, y_1, K)$. It necessarily be that G_2 detects the 21-TP after creating $(1, x_1, y_1, \perp)$ in some PROCESSSHOOT-call. By Proposition 7 (ii) and the code of PROCESSSHOOT, we know that in this call, before creating $(1, x_1, y_1, \perp)$, G_2 necessarily created another 1-query $(1, x_1', y_1', \rightarrow, n_1')$ with $y_1' = y_1 \oplus k' \oplus k''$ for $k', k'' \in \mathcal{Z}$. On the other hand, by Proposition 4 we know the 2-query $(2, x_2, y_2, n_2)$ $(x_2 = y_1' \oplus k)$ involved in the purported 21-TP was necessarily created in an earlier cycle. Thus $n_1' > n_2$ while $y_1' \oplus x_2 = k \oplus k' \oplus k'' \in 3\mathcal{Z}$, contradicting Inv2. Thus the impossibility.

Case 4: A call to PROCESSSHOOT. Assume that this call is due to G_2 popping a shoot $(i, z, \{K, K'\})$, and wlog assume that this call is made in a long simulator cycle due to D querying $\mathrm{P1}^{-1}(y_1^\circ) \rightarrow x_1^\circ$.

Now if x_1 lies in the old E-chain of the PROCESSSHOOT-call for $(i, z, \{K, K'\})$, then by the pseudocode around *NewDUShootSet*, $\mathrm{DAWARENESS}(y_3, Y3)$ should have been 0 after G_2 creating (K, x_1, y_3) and never turns 1 by Proposition 9, contradicting our assumption. On the other hand, if $x_1 \in EB(z)$, then x_1 is also in $EB(x_1^\circ)$, and the PROCESSSHOOT-call right before $x_1 \in EB(x_1^\circ)$ holds is also made in the simulator cycle due to D querying $\mathrm{P1}^{-1}(y_1^\circ)$ (otherwise $x_1 \in EB(z)$ cannot hold by Lemma 4). In this case, it holds $(i, z, \{K, K'\}) \equiv (1, x_1, \{K, K'\})$; moreover, by Lemma 4 (i), x_1 is in the new E-chain of a PROCESSSHOOT-call for a shoot $(j, z', \{K'', K'''\})$ processed earlier in this cycle. Then it can be deduced that $\mathrm{DAWARENESS}(x_1, X1)$ cannot be 0 right after $x_1 \in EB(x_1^\circ)$ holds. More clearly:

- If $K \neq K' \neq K'' \neq K'''$, then it has to be $(i, z, \{K, K'\}) = (1, x_1, \{K, K'\})$, i.e. G_2 detects (and later processes) $(1, x_1, \{K, K'\})$ after creating $(1, x_1, y_1)$. However, if $\mathrm{DAWARENESS}(x_1, X1) = 0$ then $(1, x_1, y_1)$ forming new 11-shoot contradicts Proposition 8;
- Otherwise, wlog assume $K = K'''$, then by Proposition 8, $\mathrm{DAWARENESS}(x_1, X1) = 0$ and $(i, z, \{K, K'\}) = (1, x_1, \{K, K'\})$ cannot simultaneously hold either. However, if $(i, z, \{K, K'\}) = (3, \overline{y_3}, \{K, K'\})$ with $\overline{y_3} = ETable[K](x_1)$, then there exists two 3-queries $(3, \overline{x_3'}, \overline{y_3'})$ and $(3, \overline{x_3''}, \overline{y_3''})$, and after G_2 creating $(3, \overline{x_3}, \overline{y_3}, \perp)$, it holds $\overline{x_3'} = \overline{x_3} \oplus k \oplus k''$ and $\overline{x_3''} = \overline{x_3} \oplus k \oplus k'$ (so that G_2 detects $(3, \overline{y_3}, \{K, K'\})$). But these imply $\overline{y_3'} \oplus \overline{y_3''} = k' \oplus k''$, and by an argument similar to Lemma 8 we got $(3, \overline{x_3''}, K'') \in \mathit{Completed}$ and $\overline{y_3''} \in ETable[K'']^{-1}$ before the cycle, thus $\mathrm{DAWARENESS}(x_1, X1) = 1$ right after $x_1 \in EB(x_1^\circ)$ holds.

The above exclude all possibilities and conclude. □

Proposition 12. *Since being created, an E-chain never turns bad.*

Proof. There are two possibilities for a good E-chain to turn to bad:

(i) First, in this E-chain, there might be some node x_1 (this is wlog) with $\mathrm{DAWARENESS}(x_1, X1) = 0$ and $x_1 \notin ETable[K]$ for some K, and later an E-query $(K, x_1, y_3, \rightarrow)$ is created, after which $\mathrm{DAWARENESS}(x_1, X1)$ remains 0 while $\mathrm{DAWARENESS}(y_3, Y3) = 1$;
(ii) Second, at some point the DAWARENESS functions values of some nodes in this E-chain are "flipped", after which the E-chain turns bad.

The first possibility has been excluded by Proposition 11, thus we focus on excluding the second possibility. We note that the DAWARENESS function values of the nodes of an E-chain can be flipped in the following three cases:

Case 1: D querying E *or* E^{-1}. In this case, for some x_1 with $\mathrm{DAWARENESS}(x_1, X1) = 0$, only if x_1 is adjacent to some y_3 such that $\mathrm{DAWARENESS}(y_3, Y3) = 1$ can the action turns $\mathrm{DAWARENESS}(x_1, X1)$ to 1. To show this, wlog consider D querying $\mathrm{E}^{-1}(K, y_3)$. If the E-chain contains y_3, then the claim clearly holds. Otherwise, for convenience of notations we re-assume D querying $\mathrm{E}^{-1}(K', y_3')$, then it necessarily be: (i) $x_1' = ETable[K]^{-1}(y_3')$; (ii) there exists a tuple $(1, \{(x_1, y_1), (x_1', y_1')\}) \in \mathit{DUShoots}$ before the query, and this tuple is removed after the query; (iii) $\exists (K', k') \in \mathit{HQueries} : y_1 \oplus y_1' = k \oplus k'$. By Proposition 7 (i), there exists $(K, y_3, x_1) \in \mathit{EQueries}$. By the code, it's not hard to see that the two 3-queries adjacent to y_3 and y_3' also form a shoot, and $(3, y_3, \{K_1, K_2\}) \equiv (1, x_1, \{K_1, K_2\})$. Thus it cannot be $(3, \{(\cdot, y_3), (\cdot, y_3')\}) \in \mathit{DUShoots}$ before the query, as otherwise D would have aborted in CheckDUnaware$(y_3', Y3)$. This implies DAwareness$(y_3, Y3) = 1$. Thus the claim on DAwareness$(y_3, Y3)$ holds.

Moreover, for a fixed E-chain containing x_1, only one node in this chain (say, x_1) has the DAwareness function value influenced by such an action. Formally speaking, the nodes $x_1, \ldots, x_l$ such that DAwareness$(x_i, X1) = 0$ before this action while DAwareness$(x_i, X1) = 1$ after it are not in the same E-chain. To show this, note that it's the subsequent call to RemoveDUShoots$(1, x_1)$ that flip DAwareness$(x_1, X1)$ from 0 to 1. By the code of RemoveDUShoots, it only removes two queries from $DUShoots$, i.e. $(1, x_1, y_1)$ and $(1, x_1', y_1')$, with $y_1 \oplus y_1' = k \oplus k'$ for some $k, k' \in \mathcal{Z}$. By the code, these two queries are necessarily created in an earlier ProcessShoot-call. Thus by Lemma 4 (i), x_1 and x_1' are never in the same connected component in EB. By the above, for a fixed E-chain, D querying E, etc. at most turns one of its nodes from "D-unaware" to "D-aware", and this node has to be adjacent to some "D-aware" nodes.

Case 2: D querying P2 *or* $\mathrm{P2}^{-1}$. Similarly to *Case 1*:

(i) For some x_1 with DAwareness$(x_1, X1) = 0$, only if x_1 is adjacent to some y_3 such that DAwareness$(y_3, Y3) = 1$ can the action turns DAwareness$(x_1, X1)$ to 1. To this end, wlog consider D querying $\mathrm{P2}^{-1}(y_2)$, and assume that for the following path (note that if RemoveDUShoots is called in $\mathrm{P2}^{-1}(y_2)$ and affects x_1, then x_1 and y_2 are necessarily in the same completed paths by Inv6)

$$(K, k), (K, x_1, y_3), (1, x_1, y_1), (2, x_2, y_2), (3, x_3, y_3),\ y_1 \oplus x_2 = y_2 \oplus x_3 = k \in \mathcal{Z}$$

it holds DAwareness$(x_1, X1) = 0$ before this query. Then it necessarily be DAwareness$(y_3, Y3) = 1$, as otherwise $y_2 = x_3 \oplus k$ would have caused G_2 abort in CheckDUnaware$(y_2, Y2)$ upon D querying $\mathrm{P2}^{-1}(y_2)$;

(ii) The subsequent call to RemoveDUShoots flips the DAwareness function value for at most one node per E-chain.

Case 3: G_2 processing a ProcessShoot*-call.* Informally speaking, a ProcessShoot-call causes $|DUShoots|$ decrease when the old E-chain "extends" into a shoot in $DUShoots$. This indicates that when evaluating along this old E-chain, G_2 obtains a vertex z_u in an E-chain created by an earlier ProcessShoot-call, and DAwareness$(z_u, tag) = 0$ (for the appropriate tag; the same for those below).

As the formal argument is expected to be very long and consisting of a lot of case-studies, we only give a somewhat informal presentation. Assume that:

(i) The RemoveDUShoots-call that turns DAwareness(z_u, tag) to 1 is made in a ProcessShoot-call for a shoot $(i, z, \{K_1, K_2\})$, and assume that before this call is made, all the E-chains are good. Let $z' = P_1^{-1}(k_1 \oplus k_2 \oplus P_1(z))$. The assumption of goodness of all E-chains clearly holds for the first ProcessShoot-call, and is preserved as we will demonstrate;

(ii) The E-chain containing the vertex z_u is created in the ProcessShoot-call corresponding to a shoot $(i^*, z^*, \{K_1^*, K_2^*\})$, and $z^{**} = P_1^{-1}(k_1^* \oplus k_2^* \oplus P_1(z^*))$, and $z_u \in EB(z^\circ)$ with $z^\circ \in \{z^*, z^{**}\}$.

We now focus on the point right before the ProcessShoot-call for $(i, z, \{K_1, K_2\})$ is made. Assume that the children of z_u in $Tr(z_u)$ are $z_1, \ldots, z_l$, and assume $z^\circ \in SubT(Tr(z_u), z_1)$ – informally, the E-chain between z° and z_u is of the form $z^\circ - \ldots - z_1 - z_u$. Then as DAwareness$(z^\circ, tag) = 1$ by Proposition 8 while DAwareness$(z_u, tag) = 0$, DAwareness necessarily return 0 on all the nodes in $SubT(Tr(z_u), z_2), \ldots, SubT(Tr(z_u), z_l)$ (otherwise contradicting the assumption that all the E-chains are good now).

Then, DAwareness(z', tag) also equals 1 by Proposition 8. Thus it cannot be $z' \in SubT(Tr(z_u), z_i)$ for $i = 2, \ldots, l$, otherwise contradicting the assumption that all the E-chains are good now. Thus the only possibility is $z' \in SubT(Tr(z_u), z_1)$. Additionally, the path between z_u and z' necessarily existed before the ProcessShoot-call for $(i, z, \{K_1, K_2\})$, otherwise G_2 cannot "reach" z_u by Proposition 2. Therefore, right before the RemoveDUShoots-call turning DAwareness(z_u, tag) to 1, DAwareness(z_1, tag) must already be 1. This implies that DAwareness returns 1 for all the nodes in the E-chain between z_1 and z°. As DAwareness(z_u, tag) turns 1 after the ProcessShoot-call for $(i, z, \{K_1, K_2\})$, the goodness of all E-chains are kept before and after this ProcessShoot-call.

Similarly to *Case 1*, each subsequent call to RemoveDUShoots turns at most one node per E-chain from "D-unaware" to "D-aware". These complete the analysis of *Case 3*. □

Lemma 9. *During any execution D^{G_2}, all E-chains are good.*

Proof. Simply gathering Propositions 10 and 12. □

For an E-chain, if each of its nodes has the DAWARENESS function value equals 1, then this chain is called *D-aware*. At the end of each chain-reaction call, as long as G_2 does not abort, the length of D-aware alternated E-chains cannot exceed the total number of E- and P-cycle (cf. subsection 7.1). The proof relies on two sub-claims as follows.

Proposition 13. *In any simulator cycle, at the end of each chain-reaction call, as long as G_2 does not abort, the length of any* D-aware *E-chain newly created in this cycle does not exceed the number of E- and P-cycles that have happened before.*

Proof. We consider the cycle in which the first E-query (K_1, x_1, y_3) of a D-aware E-chain is created. Here by "created" we mean the creation of the first E-query (K_1, x_1, y_3) of this chain with DAWARENESS$(x_1, X1) =$ DAWARENESS$(y_3, Y3) = 1$. Note that (K_1, x_1, y_3) may not be "really" created in this cycle: it may already existed, but it is this cycle that flips DAWARENESS$(x_1, X1)$ or DAWARENESS$(y_3, Y3)$ (or both) from 0 to 1.

We make discussion for each cycle as follows:

Case 1: A cycle due to D querying H, P1, *or* P3^{-1}. During such a cycle, it's not hard to see: (a) no new E-query is created; (b) $|DUShoots|$ does not decrease. Thus such a cycle cannot "create" any new D-aware E-chains.

Case 2: A cycle due to D querying E *or* E^{-1}. Wlog consider D querying $\mathrm{E}(K, x_1) \to y_3$. It has to be DAWARENESS$(x_1, X1) = 1$. We show that the newly "created" D-aware E-chain $x_1 - y_3$ has length at most 1. For this we distinguish two sub-cases:

(i) $x_1 \notin ETable[K]$ before the query. Then by Inv4, right after this cycle, it holds $\forall K' \neq K, y_3 \notin ETable[K']^{-1}$. Thus the newly created E-chain $x_1 - y_3$ has length 1;

(ii) $x_1 \in ETable[K]$ before the query, say, this cycle triggers a call to REMOVEDUSHOOTS$(3, y_3)$, which turns DAWARENESS$(y_3, Y3)$ as well as DAWARENESS$(y_3', Y3)$ for another node y_3' from 0 to 1. According to our assumption, x_1 is the only node of the imagined E-chain that has its DAWARENESS function value equals 1. On the other hand, y_3 and y_3' cannot be in the same E-chain, cf. the analysis in *Case 1* of Proposition 12. Thus y_1 is the only node of the imagined E-chain that have its DAWARENESS function value "flipped" during this cycle, and thus the newly created D-aware E-chain $x_1 - y_3$ has length 1.

Clearly, at least one E-/P-cycle (i.e. the cycle for $\mathrm{E}(K, x_1)$) has happened. Thus in this case, the length of "newly created" D-aware E-chains does not exceed the number of earlier E- and P-cycles.

Case 3: A cycle due to D querying P2 *or* P2^{-1}. Wlog consider D querying $\mathrm{P2}(x_2) \to y_2$. We also show that the newly "created" D-aware E-chain $x_1 - y_3$ has length at most 1. For this we distinguish three sub-cases:

(i) $x_2 \notin P_2$ before the query, and $\nexists k \in \mathcal{Z} : x_2 \oplus k \in P_1^{-1}$. Then by the code, (a) no new E-query is created; (b) $|DUShoots|$ does not decrease.

(ii) $x_2 \notin P_2$ before the query, and $\exists k \in \mathcal{Z} : x_2 \oplus k \in P_1^{-1}$. By the code, if G_2 does not abort, then there exists exactly one $(K, k) : x_2 \oplus k \in P_1^{-1}$. Let the 1-query adjacent to $x_2 \oplus k$ be $(1, x_1, y_1)$. According to the code, if this cycle "creates" a new D-aware E-chain, then it has to be $x_1 \notin ETable[K]$ (and thus G_2 creates a new E-query $(K, x_1, y_3, \to)$).
Then the case is similar to *Case 2* (i): (a) DAWARENESS$(x_1, X1) = 1$, otherwise CHECKDUNAWARE$(x_2, X2)$ would have caused abort; (b) right after this cycle, it holds $\forall K' \neq K, y_3 \notin ETable[K']^{-1}$ by Inv4, and thus the newly created E-chain $x_1 - y_3$ has length 1.

(iii) $x_2 \in P_2$ before the query, say, this cycle triggers a call to REMOVEDUSHOOTS$(3, y_3)$ for some y_3. Then the case is similar to *Case 2* (ii), and the length of the "newly created" D-aware E-chain is at most 1.

Thus in this case, the length of "newly created" D-aware E-chains does not exceed the number of earlier E- and P-cycles either.

Case 4: A cycle due to D querying P1^{-1} *or* P3. Wlog consider D querying $\mathrm{P1}^{-1}(y_1^\circ) \to x_1^\circ$. If $y_1^\circ \in P_1^{-1}$ before the cycle, then (similarly to *Case 1*): (a) no new E-query is created; (b) $|DUShoots|$ does not decrease. Thus we focus on the case of $y_1^\circ \notin P_1^{-1}$, i.e. the case of a long simulator cycle.

Assume that in this cycle, l E-queries either are newly created or have their corresponding DAWARENESS function values "flipped", and form a D-aware (K_1, K_2)-alternated E-chain with length l. To show the main claim, *we associate a unique earlier E-/P-cycle to each of them.* Consider one of them, e.g. (K_1, x_1, y_3, d_1). The action around this query may be due to two possibilities:

Sub-case 4.1: (K_1, x_1, y_3) is a newly created query. We further distinguish two cases:

Sub-case 4.1.1: (K_1, x_1, y_3) is created in a call to Process21TP(x_1, y_1, K). Let the involved 2-query be $(2, x_2, y_2)$ ($x_2 = y_1 \oplus k$). Then by Proposition 4, this 2-query was necessarily created in an earlier cycle due to D querying $\mathrm{P2}(x_2)$ or $\mathrm{P2}^{-1}(y_2)$. Furthermore, if two different such E-queries $(K_1, x_{1,i}, y_{3,i})$ and $(K_1, x_{1,j}, y_{3,j})$ are associated with the same 2-query $(2, x_2, y_2)$, then $(2, x_2, y_2)$ is involved in two distinct MidTPs, contradicting Proposition 5. Thus each E-query created in Process21TP-calls is associated with a unique earlier cycle due to D querying $\mathrm{P2}(x_2)$ or $\mathrm{P2}^{-1}(y_2)$. The case of (K_1, x_1, y_3) created in a call to Process23TP is similar by symmetry.

Sub-case 4.1.2: (K_1, x_1, y_3) is created in a ProcessShoot*-call corresponding to G_2 popping $(i', z, \{K_1, K'\})$.* Wlog assume $d_1 = \rightarrow$. Then it has to be $x_1 \in EB(z)$ and thus $x_1 \in EB(x_1^\circ)$, as otherwise the fact that $x_1 \notin ETable[K_1]$ would have caused G_2 adding the shoot containing y_3 to $DUShoots$, so that DAwareness$(y_3, Y3)$ equals 0 and cannot be flipped in this cycle due to Proposition 9, a contradiction. Thus there exists some E-query (K', x_1', y_3') which existed before this ProcessShoot-call, and in this call, when G_2 is evaluating along the old E-chain, it reaches (K', x_1', y_3'), finds $x_1' \in ETable[K']$, and thus does not add the shoot containing y_3 to $DUShoots$.

We now show that two different such new E-queries $(K_i, x_{1,i}, y_{3,i})$ and $(K_j, x_{1,j}, y_{3,j})$ cannot be associated with the same pre-existing E-query (K', x_1', y_3').[12] By the pseudocode of ProcessShoot, if this situation occurs, then it would hold $P_1(x_{1,i}) \in B_2(y_1^\circ)$ and $P_1(x_{1,i}) \in B_2(y_1^\circ)$. Then $P_1(x_{1,i}) = P_1(x_1') \oplus k_i \oplus k'$ and $P_1(x_{1,j}) = P_1(x_1') \oplus k_j \oplus k'$ implies $P_1(x_{1,i}) \oplus P_1(x_{1,j}) = k_i \oplus k_j \in 2\mathcal{Z}$, the existence of a pseudo-cycle similar to that appeared in the proof of Proposition 9. Thus two different such new E-queries $(K_i, x_{1,i}, y_{3,i})$ and $(K_j, x_{1,j}, y_{3,j})$ are associated with two different pre-existing E-queries $(K_i', x_{1,i}', y_{3,i}')$ and $(K_j', x_{1,j}', y_{3,j}')$.

Now, we argue that $(K_i', x_{1,i}', y_{3,i}')$ and $(K_j', x_{1,j}', y_{3,j}')$ must be created in two different earlier E-/P-cycles. For this we consider G_2 creating $(K_i', x_{1,i}', y_{3,i}')$ and $(K_j', x_{1,j}', y_{3,j}')$. Cf. *Case 1* of this proof, this cannot be due to D querying H, P1, or $\mathrm{P1}^{-1}$. Thus this may be due to the following possibilities:

- D querying E, E^{-1}, P2, or $\mathrm{P2}^{-1}$. It's not hard to see that each such cycle creates at most *1* E-queries. Thus if $(K_i', x_{1,i}', y_{3,i}')$ and $(K_j', x_{1,j}', y_{3,j}')$ are both created in such cycles, then they would have two different associated cycles;
- A long cycle due to e.g. D querying $\mathrm{P1}^{-1}(y_1^*)$. Let $y_{1,i}' = P_1(x_{1,i}')$ and $y_{1,j}' = P_1(x_{1,j}')$. Then by the code, we know that in the later ProcessShoot-call, G_2 creates two AD-1-queries $(1, x_{1,i}, y_{1,i}, \perp)$ and $(1, x_{1,j}, y_{1,j}, \perp)$, with $y_{1,i} = y_{1,i}' \oplus k_i \oplus k_i'$ and $y_{1,j} = y_{1,j}' \oplus k_j \oplus k_j'$. Now, in the earlier ProcessShoot-call,
  - if G_2 creates two AD-1-queries $(1, x_{1,i}', y_{1,i}', \perp)$ and $(1, x_{1,j}', y_{1,j}', \perp)$, then by Lemma 4 (ii), it holds $y_{1,i}', y_{1,j}' \in B_2(y_1^*)$. This along with $y_{1,i}, y_{1,j} \in B_2(z)$ indicates the existence of a "pseudo-cycle" $z - \ldots - y_{1,i} \xrightarrow{\oplus k_i \oplus k_i'} y_{1,i}' - \ldots - y_1^* - \ldots - y_{1,j}' \xrightarrow{\oplus k_j \oplus k_j'} y_{1,j} - \ldots - (z)$ in B_2, cf. Fig. 6 (left), which would ultimately contradict Lemma 6 (similarly to Proposition 5, albeit more complicated).
  - if G_2 does not create $(1, x_{1,i}', y_{1,i}', \perp)$ nor $(1, x_{1,j}', y_{1,j}', \perp)$, then there exists $k_i'', k_j'' \in \mathcal{Z}$ such that G_2 creates two AD-1-queries $(1, x_{1,i}'', y_{1,i}'', \perp)$ and $(1, x_{1,j}'', y_{1,j}'', \perp)$ with $y_{1,i}'' = y_{1,i}' \oplus k_i' \oplus k_i''$ and $y_{1,j}'' = y_{1,j}' \oplus k_j' \oplus k_j''$. We also have $y_{1,i}'', y_{1,j}'' \in B_2(y_1^*)$ by Lemma 4 (ii). This along with $y_{1,i}, y_{1,j} \in B_2(z)$ indicates the existence of a "pseudo-cycle" $z - \ldots - y_{1,i} \xrightarrow{\oplus k_i \oplus k_i''} y_{1,i}'' - \ldots - y_1^* - \ldots - y_{1,j}'' \xrightarrow{\oplus k_j \oplus k_j''} y_{1,j} - \ldots - (z)$ in B_2, cf. Fig. 6 (right), which would ultimately contradict Lemma 6.
  - The "hybrid case": if G_2 creates $(1, x_{1,i}', y_{1,i}', \perp)$ but not $(1, x_{1,j}', y_{1,j}', \perp)$, then following the same line as the above discussion, it can be seen that a "pseudo-cycle" $z - \ldots - y_{1,i} \xrightarrow{\oplus k_i \oplus k_i'} y_{1,i}' - \ldots - y_1^* - \ldots - y_{1,j}'' \xrightarrow{\oplus k_j \oplus k_j''} y_{1,j} - \ldots - z$ would be in B_2.

The "pseudo-cycle" appeared in the above discussion implies that, among the two ProcessShoot-calls that create $(K_i', x_{1,i}', y_{3,i}')$ and $(K_j', x_{1,j}', y_{3,j}')$, the later one necessarily causes abort before it returns.[13]Whereas the premise of this lemma is *all the earlier chain-reaction calls returned without abortion.* Thus the analysis for *sub-case 4.1.2* is completed.

[12] $(K_i, x_{1,i}, y_{3,i})$ and $(K_j, x_{1,j}, y_{3,j})$ may be created in two different ProcessShoot-calls. But this does not affect the agrement.

[13] If $(K_i', x_{1,i}', y_{3,i}')$ and $(K_j', x_{1,j}', y_{3,j}')$ are created in the same ProcessShoot-call, then this call necessarily aborts.

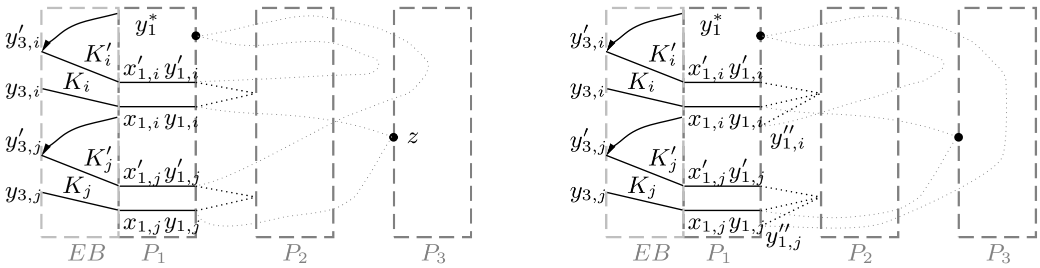


Fig. 6. For Proposition 13: the two "pseudo-cycles" for *sub-case 4.1.2*. The two arrowed curves indicate G_2 evaluating along the old E-chains in the later PROCESSSHOOT-call.

Sub-case 4.2: (K_1, x_1, y_3) is included due to a call PROCESS11SHOOT(x'_1, y'_1, K_1, K') *flipping the* DAWARENESS *function value of x_1 or y_3.* This sub-case describes one of the following two cases:

(i) G_2 reaching the query (K_1, x_1, y_3) when evaluating along the old E-chain of the PROCESSSHOOT-call for $(i', z, \{K_1, K'\})$, and thus calling REMOVEDUSHOOTS and removing e.g. a tuple containing y_3 from *DUShoots*;
(ii) G_2 reaching a query (K'_1, x'_1, y'_3) when evaluating along the old E-chain of the PROCESSSHOOT-call for $(i', z, \{K_1, K'\})$, and calling REMOVEDUSHOOTS and removing e.g. a tuple of the form $(3, \{(\cdot, y_3), (\cdot, y'_3)\})$ from *DUShoots*.

By Proposition 9, in each case, (K_1, x_1, y_3) was necessarily created in an earlier long cycle; thus by Lemma 4 (i), it holds $x_1 \notin EB(x^\circ_1)$.

Now, similarly to *sub-case 4.1.2* above, we argue that two such "flipped" E-queries $(K'_i, x'_{1,i}, y'_{3,i})$ and $(K'_j, x'_{1,j}, y'_{3,j})$ must be created in two different earlier E-/P-cycles. First, wlog assume that DAWARENESS$(y'_{3,i}, Y3) =$ DAWARENESS$(y'_{3,j}, Y3) = 1$ before this call, and this call flips the DAWARENESS function value of $x'_{1,i}$ and $x'_{1,j}$. Then:

(i) If $x'_{1,i}$ and $x'_{1,j}$ are not in the same shoot, then the analysis follows the same line as *sub-case 4.1.2* – specifically, leading to "pseudo-cycles" in each case. This implies that among the two PROCESSSHOOT-calls that flip DAWARENESS$(x'_{1,i}, X1)$ and DAWARENESS$(x'_{1,j}, X1)$, the later one necessarily aborts (similarly to *sub-case 4.1.2*, if they are flipped in the same call, then this call would abort);
(ii) If $x'_{1,i}$ and $x'_{1,j}$ are in the same shoot, then by Lemma 4 and Proposition 2, $(K'_i, x'_{1,i}, y'_{3,i})$ and $(K'_j, x'_{1,j}, y'_{3,j})$ cannot be in the same connected component.

This also shows that for (K_1, x_1, y_3), it cannot be DAWARENESS$(x_1, X1) =$ DAWARENESS$(y_3, Y3) = 0$ before this cycle while DAWARENESS$(x_1, X1) =$ DAWARENESS$(y_3, Y3) = 1$ after this cycle: because such a query (K_1, x_1, y_3) was necessarily created in an earlier PROCESSSHOOT, and thus adjacent to some (K_1, K')-alternated E-chain. Hence DAWARENESS$(x_1, X1) =$ DAWARENESS$(y_3, Y3) = 1$ after this cycle would contradict what we have just argued. By the above, each such increment in length can also be associated with a unique earlier E-/P-cycle. These complete the analysis for *sub-case 4.2*.

Summary for Case 4. By the above, all the "newly created" E-queries are in $EB(x^\circ_1)$, while all the "flipped" E-queries are *not* in $EB(x^\circ_1)$. Thus by Proposition 2, the two types of new "D-aware" E-queries do not add up. As the number of each type does not exceed the number of earlier E- and P-cycles, we reach the claim, and complete the proof. ☐

Proposition 14. *For any fixed (K_1, K_2)-alternated E-chain, since being created, its length increases by at most 1 after each E- and P-cycle, while stays constant during H-cycles.*

Proof. Similarly to Proposition 12, there are also two possibilities for such an E-chain extending:

(i) First, in this E-chain, there might be some node x_1 with $x_1 \notin ETable[K_1]$ (wlog), and later an E-query $(K_1, x_1, y_3, \rightarrow)$ is created with DAWARENESS$(y_3, Y3) = 1$;
(ii) Second, at some point the DAWARENESS function values of some nodes in this E-chain are "flipped'.

We make discussion for each cycle as follows:

Case 1: A cycle due to D querying H*,* P1*, or* $\mathrm{P3}^{-1}$*.* As discussed in the proof of Proposition 13, in such a cycle no new E-query is created and no node has its DAwareness function value flipped. Thus the length of each pre-existing D-aware E-chain stays constant.

Case 2: A cycle due to D querying E *or* E^{-1}*.* Wlog consider D querying $\mathrm{E}(K, x_1)$. If $x_1 \notin ETable[K]$, then this cycle may bring new E-query to pre-existing E-chains. By Inv4, $y_3 \notin ETable[K']$ holds for any $K' \neq K$, and thus the increment is at most 1. On the other hand, if $x_1 \in ETable[K]$, then this cycle may flip some DAwareness function values. However, Cf. the analysis in *Case 1* of Proposition 12, for a fixed E-chain, D querying $\mathrm{E}(K, x_1)$ turns at most one of its nodes from "D-unaware" to "D-aware". Thus such increment does not exceed 1 either.

Case 3: A cycle due to D querying P2 *or* $\mathrm{P2}^{-1}$*.* Wlog consider D querying $\mathrm{P2}(x_2)$. We distinguish three sub-cases similar to *Case 3* of Proposition 13:

(i) $x_2 \notin P_2$, and $\nexists k \in \mathcal{Z} : x_2 \oplus k \in P_1^{-1}$. Then no new E-query is created, and $|DUShoots|$ does not decrease, thus no increment.
(ii) $x_2 \notin P_2$, and $\exists k \in \mathcal{Z} : x_2 \oplus k \in P_1^{-1}$, and $x_1 \notin ETable[K]$ for $x_1 = P_1^{-1}(x_2 \oplus k)$, so that a new E-query is created in this cycle. Then the case is similar to *Case 2*: by Inv4, $y_3 \notin ETable[K']$ holds for any $K' \neq K$, and thus the increment is at most 1.
(iii) $x_2 \in P_2$, say, this cycle triggers a call to RemoveDUShoots$(3, y_3)$ for some y_3. Then Cf. the analysis in *Case 1* of Proposition 12, for a fixed E-chain, D querying P2 turns at most one of its nodes from "D-unaware" to "D-aware". Thus in this case, the increment does not exceed 1 either.

Case 4: A (long) cycle due to D querying $\mathrm{P1}^{-1}$*, or* P3*.* We first argue that long cycles cannot bring in "new-E-query-type" increment to pre-existing E-chains. Wlog consider D querying $\mathrm{P1}^{-1}(y_1^\circ) \to x_1^\circ$. We note that (K, x_1, y_3) cannot be created due to G_2 subsequent processing a MidTP (i, z, K), as $x_1 \notin EB(x_1^\circ)$ by Lemma 4 (i). Thus x_1 lies in the old E-chain of a subsequent ProcessShoot-call. As $x_1 \notin ETable[K]$, after this ProcessShoot-call, it would hold DAwareness$(y_3, Y3) = 0$, which would not be flipped in this cycle due to Proposition 9.

By this, pre-existing D-aware E-chains extend only due to subsequent calls to RemoveDUShoots. As RemoveDUShoots may be called more than once, the case is more complicated than *Case 2* and *3*. However, we proceed to show that the length of any alternated E-chain cannot increase by more than 1. To this end, we make the following assumptions:

(i) There exist four 1-queries $(1, x_1, y_1, \to)$, $(1, x_1', y_1', \perp)$, $(1, x_1'', y_1'', \to)$, and $(1, x_1''', y_1''', \perp)$ with $y_1 \oplus y_1' = y_1'' \oplus y_1''' = k_1 \oplus k_2$, DAwareness$(x_1, X1) = 0$, and DAwareness$(x_1'', X1) = 0$. The soundness of this assumption comes from Proposition 7 and the code of ProcessShoot;
(ii) $x_1'' = xebval_l(K_1, K_2, x_1)$, say, the two "D-unaware" shoots are in the same (K_1, K_2)-alternated E-chain;
(iii) In a long simulator cycle due to D querying $\mathrm{P1}(y_1^\circ)$, G_2 reaches first x_1 and then x_1'' when evaluating along the old E-chains in subsequent ProcessShoot-calls, which causes both $(1, \{(x_1, y_1), (x_1', y_1')\})$ and $(1, \{(x_1'', y_1''), (x_1''', y_1''')\})$ be removed from *DUShoots* (and thus the length of the (K_1, K_2)-alternated E-chain underlying the two shoots increases by two).

Note that these assumptions are made concrete for clearness, but they are wlog. For example, one could substitute $(1, x_1'', y_1'', \to)$ and $(1, x_1''', y_1''', \perp)$ with $(3, x_3, y_3, \leftarrow)$ and $(3, x_3', y_3', \perp)$, and the argument carries as well.

Now, assume that after G_2 reaches x_1, it is to create an AD-1-query of the form $(1, \cdot, y_1 \oplus k_1 \oplus k_3)$; after G_2 reaches x_1'', it is to create an AD-1-query of the form $(1, \cdot, y_1'' \oplus k_1 \oplus k_4)$, cf. Fig. 7 (according to the pseudocode, this assumption is reasonable). Then by Lemma 4 (ii), right before G_2 calls RemoveDUShoots$(y_1'' \oplus k_1 \oplus k_4, Y1)$ (and creates $(1, \cdot, y_1'' \oplus k_1 \oplus k_4)$), it holds:

- $y_1 \oplus k_1 \oplus k_3 \in B_2(y_1^\circ)$ and $y_1'' \oplus k_1 \oplus k_4 \in B_2(y_1^\circ)$;
- there exists z such that $y_1' \in B_2(z)$ and $y_1''' \in B_2(z)$ (because $(1, x_1', y_1', \perp)$ and $(1, x_1''', y_1''', \perp)$ are created in the same ProcessShoot-call).

Note that $y_1 \oplus k_1 \oplus k_3 = y_1' \oplus k_2 \oplus k_3$ and $y_1'' \oplus k_1 \oplus k_4 = y_1''' \oplus k_2 \oplus k_4$. This implies a "pseudo-cycle" $y_1 - \ldots - y_1''' \xrightarrow{\oplus k_2 \oplus k_4} y_1'' \oplus k_1 \oplus k_4 - \ldots - y_1 \oplus k_1 \oplus k_3 \xrightarrow{\oplus k_2 \oplus k_3} (y_1)$ in B_2, cf. Fig. 7 (right), which would ultimately contradict Lemma 6. Thus G_2 should have aborted, and would not call RemoveDUShoots$(y_1'' \oplus k_1 \oplus k_4, Y1)$ to remove the second shoot.

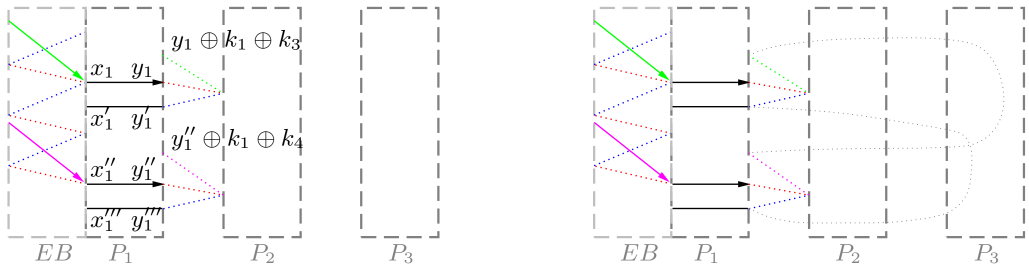


Fig. 7. For Proposition 14, *Case 4*: lines in blue and red indicate the (K_1, K_2)-alternated E-chain, while lines in green and magenta indicate E-queries with keys K_3 and K_4, resp. (right) illustration for the "pseudo-cycle".

The above discussion assumes x_1 and x_1'' in the old E-chain of the earlier ProcessShoot-call. If not, i.e. the four involved 1-queries are $(1, x_1, y_1, \perp)$, $(1, x_1', y_1', \rightarrow)$, $(1, x_1'', y_1'', \perp)$, and $(1, x_1''', y_1''', \rightarrow)$ (with the *dir* values "swapped"), then the pseudo-cycle $y_1 - \ldots - y_1'' \xrightarrow{\oplus k_1 \oplus k_4} y_1'' \oplus k_1 \oplus k_4 - \ldots - y_1 \oplus k_1 \oplus k_3 \xrightarrow{\oplus k_1 \oplus k_3} y_1$ still exists. These complete the proof. $\square$

Gathering the above yields the desired claim.

Lemma 10. *At the end of each chain-reaction call, if G_2 does not abort, then for any (K_1, K_2), the length of D-aware (K_1, K_2)-alternated E-chain is at most $q_e + q_p$.*

Proof. By Propositions 13 and 14, the length of any D-aware (K_1, K_2)-alternated E-chain does not exceed the total number of E- and P-cycles, which does not exceed $q_e + q_p$ and thus enforcing the claimed bound. Note that although Proposition 14 holds "unconditionally", Proposition 13 enforces the condition(s) of this lemma. $\square$

Shoots in *Border* can never be "reached" by D.

Lemma 11. *For any tuple $(1, \{(x_1, y_1), (x_1', y_1')\}) \in DUShoots$, (a) $x_1 \in Border \Leftrightarrow x_1' \in Border$; (b) if $x_1 \in Border$ then $(1, \{(x_1, y_1), (x_1', y_1')\}) \in DUShoots$ always holds.*

Proof. $x_1 \in Border \Leftrightarrow x_1' \in Border$ can be seen from the code of Process11Shoot. On the other hand, if (b) does not hold, then there are two possibilities:

(i) In some Process11Shoot-call, a shoot formed at the "endpoints" is not added to *DUShoots*;
(ii) For some $x_1 \in Border$, the shoot $(1, \{(x_1, y_1), (x_1', y_1')\})$ (containing it) is later removed from *DUShoots*.

Consider possibility (i) first. Wlog assume that in Process11Shoot(x_1^*, y_1^*, K_1, K_2) (let $x_1^{**} = P_1^{-1}(y_1^* \oplus k_1 \oplus k_2)$), when G_2 obtains $y_3' = xebval_{2t-1}(x_1^{**}, K_1, K_2)$, it finds $y_3' \in ETable[K_2]^{-1}$, and thus does not add the shoot containing $x_1' = xebval_{2t}(x_1^{**}, K_1, K_2)$ into *DUShoots*. This indicates the E-chain $x_1^{**} - \ldots - y_3' - x_1'$ exists before Process11Shoot(x_1^*, y_1^*, K_1, K_2) (otherwise $y_3' \in ETable[K_2]^{-1}$ is not possible by Proposition 2). By Proposition 8, before the call to Process11Shoot(x_1^*, y_1^*, K_1, K_2), it already holds DAwareness$(x_1^{**}, X1) = 1$. According to how (K_2, x_1', y_3') is created, we distinguish two cases:

Case 1.1: (K_2, x_1', y_3') is created in a short cycle, or a call to Process21/23TP. Then it necessarily holds DAwareness$(y_3', Y3) = 1$, thus by Lemma 9 we know before the call to Process11Shoot(x_1^*, y_1^*, K_1, K_2), the E-chain $x_1^{**} - \ldots - y_3'$ is a D-aware alternated E-chain, with length $2t - 1 > q_e + q_p$; this contradicts Lemma 10.

Case 1.2: (K_2, x_1', y_3') is created in the ProcessShoot*-call for a shoot $(i, z, \{K_2, K_3\})$.* In this case, it has to be $K_3 \neq K_1$, otherwise $(1, x_1^*, \{K_1, K_2\}) \equiv (i, z, \{K_2, K_1\})$ and thus Process11Shoot(x_1^*, y_1^*, K_1, K_2) would not happen. Wlog assume $(i, z) = (1, x_1^\circ)$, and let $x_1^{\circ\circ} = P_1^{-1}(P_1(x_1^\circ) \oplus k_2 \oplus k_3)$. Then it's not hard to see either x_1° or $x_1^{\circ\circ}$ lies in $SubT(x_1^{**}, y_3')$. By Proposition 8 we know the DAwareness function values of both x_1° and $x_1^{\circ\circ}$ are 1, thus by Lemma 9 we (again) have DAwareness$(y_3', Y3) = 1$, and before Process11Shoot(x_1^*, y_1^*, K_1, K_2), the E-chain $x_1^{**} - \ldots - y_3'$ is a D-aware alternated E-chain with length $2t - 1$, contradicting Lemma 10.

We then consider possibility (ii). Wlog assume $(1, \{(x_1, y_1), (x_1', y_1')\})$ is added to *DUShoots* in Process11Shoot(x_1^*, y_1^*, K_1, K_2), and $x_1 \in EB(x_1^*)$. Then we exclude two cases for it being removed:

Case 2.1: D querying E, E^{-1}, P2, *or* $\mathrm{P2}^{-1}$. Then it can be seen from the analysis of *Case 1* and *2* in Proposition 12 that the current cycle can only increase the length of the D-aware (K_1, K_2)-alternated E-chain containing x_1^* by at most 1. Whereas by Proposition 8 we have $\text{DAwareness}(x_1^*, X1) = 1$. Therefore, before this cycle, there necessarily exists a D-aware (K_1, K_2)-alternated E-chain with length $2t - 1 > q_e + q_p$, contradicting Lemma 10.

Case 2.2: G_2 processing a long cycle. It can be seen from the analysis of *Case 3* in Proposition 12 that long cycle can only increase the length of the D-aware (K_1, K_2)-alternated E-chain containing x_1^* by at most 1. Thus (similarly to *Case 2.1*), before this cycle, there exists a D-aware (K_1, K_2)-alternated E-chain with length $2t - 1 > q_e + q_p$, contradicting Lemma 10. These complete the proof. □

Another observation is that the old E-chains of ProcessShoot-calls never "extend into" the set *Border*.

Proposition 15. *Consider the old E-chain of a* ProcessShoots*-call. None of the values in this chain lies in the set Border, except for the two new endpoints.*

Proof. Assume otherwise, i.e. in the ProcessShoots-call corresponding to popping a shoot $(i, z, \{K_1, K_2\})$, when G_2 is evaluating along the old E-chain, it obtains a value $x_{1,2t+1}^*$ which was added to *Border* by an earlier call to $\text{Process11Shoot}(x_1^*, y_1^*, K_1^*, K_2^*)$, and $x_{1,2t+1}^* \in EB(x_1^*)$ (these are wlog). Let $z' = P_1^{-1}(k_1 \oplus k_2 \oplus P_1(z))$. Then, right before $(i, z, \{K_1, K_2\})$ is popped, the E-chain between z' and $x_{1,2t+1}^*$ exists, as otherwise $x_{1,2t+1}^* \in EB(z')$ never holds by Lemma 2 and G_2 cannot obtain $x_{1,2t+1}^*$. Then, similarly to Lemma 11, if $(K_1, y_3', x_{1,2t+1}^*)$ is created in a short cycle or a call to Process21/23TP, then it necessarily be $\text{DAwareness}(x_{1,2t+1}^*, X1) = 1$, thus before the ProcessShoot-call for $(i, z, \{K_1, K_2\})$, the E-chain $x_1^* - \ldots - x_{1,2t+1}^*$ is a D-aware alternated E-chain with length $2t > q_e + q_p$, contradicting Lemma 10. On the other hand, if $(K_1, y_3', x_{1,2t+1}^*)$ is created in $\text{Process11Shoot}(x_1^*, y_1^*, K_1^*, K_2^*)$, then it's not hard to see $z' \in SubT(x_1^*, y_3')$, thus the DAwareness function value of z' is 1 by Proposition 8, and further $\text{DAwareness}(z', Y3) = 1$ by Lemma 9. Therefore, before the ProcessShoot-call for $(i, z, \{K_1, K_2\})$ happens, the E-chain $x_1^* - \ldots - y_3'$ is a D-aware alternated E-chain with length $2t - 1 > q_e + q_p$, contradicting Lemma 10. Thus the claim. □

Remark 2. Back to the proof of Proposition 15, one may note that according to the assumption, the later ProcessShoots-call will cause the shoot in *Border* be removed from *DUShoots*, thus contradicting Lemma 11. However, we should show *the old E-chains never extend into Border*, rather than *such a* ProcessShoots*-call is deemed to abort in future*. Thus we cannot simply prove it via Lemma 11.

8 Assertions and Adaptations Never Cause Abort

With the above preparations, we are able to prove the non-abortion of assertions and adaptations in simulator cycles.

8.1 Short Simulator Cycles Can be Correctly Handled

Recall that short cycles are induced by D making P1, P2, $\mathrm{P2}^{-1}$, $\mathrm{P3}^{-1}$, or H, and are simpler to analyzed.

Lemma 12. *The adaptations and assertions in a simulator cycle induced by D making* $\mathrm{P1}(x_1)$ *or* $\mathrm{P3}^{-1}(y_3)$ *never cause abort.*

Proof. Consider the cycle due to $\mathrm{P1}(x_1)$ first. Assuming $x_1 \notin P_1$, as the other case is of no interest. If G_2 does not abort in the subsequent RandAssign-call, then right after RandAssign returns y_1, by Inv2 and Inv3 it holds

$$\forall z \in 5\mathcal{Z}, y_1 \oplus z \notin P_2 \text{ and } \forall z \in 6\mathcal{Z}, y_1 \oplus z \notin P_1^{-1}. \tag{1}$$

By construction, G_2 then complete a chain for each (K_i, k_i) and $(3, x_3^i, y_3^i)$ such that $\text{Check}(K_i, x_1, y_3^i) =$ **true** (or: the E-query $(K_i, x_1, y_3^i, edir^i, enum^i)$ pre-exists). The adaptations and assertions in these chain-completions constitute all those in this cycle.

Consider the chain-completion for (K_i, x_1, y_3^i) and $(3, x_3^i, y_3^i)$. We first prove that the adaptation does not cause abort. Consider $x_2^i = y_1 \oplus k_i$ first. By (1), $x_2^i \notin P_2$ holds right after the RandAssign-call. During the period between RandAssign and $\text{Adapt}(2, x_2^i, y_2^i, edir^i, enum^i)$, there only exist calls to $\text{Adapt}(2, x_1 \oplus k_j, \ldots)$ for $K_j \neq K_i$. As $K_j \neq K_i$ implies $k_j \neq k_i$ and $x_2^j = y_1 \oplus k_j \neq x_2^i$, these earlier Adapt-calls cannot add x_2^i to P_2. Thus $x_2^i \notin P_2$ holds till the call $\text{Adapt}(2, x_2^i, y_2^i, edir^i, enum^i)$ is made.

Consider $y_2^i = x_3^i \oplus k_i$ then. Right before the simulator cycle, $y_2^i = x_3^i \oplus k_i \in P_2^{-1}$ is not possible, as otherwise $(3, K_i, x_3^i)$ should have been in *Completed* by Inv6 and $x_1 = ETable[K_i]^{-1}(y_3^i)$ should have been in P_1, contradicting the assumption $x_1 \notin P_1$ at the beginning of the proof. Moreover y_2^i cannot be added to P_2^{-1} by the earlier ADAPT-calls. For this, consider such a call to $\text{ADAPT}(2, x_2^j, y_2^j, edir^j, enum^j)$ with $j \neq i$. Note that the involved E-queries (K_j, x_1, y_3^j) necessarily has $y_3^j \neq y_3^i$, as otherwise contradicting Inv4. The four involved queries (K_i, x_1, y_3^i), (K_j, x_1, y_3^j), $(3, x_3^i, y_3^i)$, and $(3, x_3^j, y_3^j)$ have been in the history before this cycle, and the two E-queries were live. Thus by Lemma 5, among $(3, x_3^i, y_3^i)$ and $(3, x_3^j, y_3^j)$, the one created later has direction $\leftarrow$. Thus $y_3^i \oplus k_i = x_2^i = x_2^j = y_3^j \oplus k_j$ is not possible by Inv3. Thus $y_2^i \notin P_2^{-1}$ till the call to $\text{ADAPT}(2, x_2^i, y_2^i, edir^i, enum^i)$. By the above, the call to $\text{ADAPT}(2, x_2^i, y_2^i, edir^i, enum^i)$ would not cause abort.

Then consider the subsequent assertions. The first assertion causes abort if $\exists k' \neq k_i : x_2^i \oplus k' \in P_1^{-1}$. This implies $x_1 \oplus k_i \oplus k' \in P_1^{-1}$, which is not possible right after RANDASSIGN by (1), and would never be possible during the cycle since no new 1-query would be created. The second assertion causes abort if $\exists k' \neq k_i : y_2^i \oplus k' \in P_3$. This implies $x_3^i \oplus k_i \oplus k' \in P_3$. This is not possible before the cycle. To show this, we first note that all the involved E-queries are due to D (by Proposition 3). Thus it holds $\text{DAWARENESS}(x_1, X1) = 1$ and $\forall i, \text{DAWARENESS}(y_3^i, Y3) = 1$. Thus if $x_3^i \oplus k_i \oplus k' \in P_3$, then the 33-shoot $(3, y_3^i, \{k_i, k'\})$ cannot be in *Border* by Lemma 11, and $(3, K_i, x_3^i) \in Completed$ by Inv8 and Inv7, and thus $x_1 \in P_1$, a contradiction. As no new 3-query would be created in the cycle, the second assertion would not cause abort either. These complete the proof for $\text{P1}(x_1)$.

The argument for $\text{P3}^{-1}(y_3)$ is similar by symmetry. □

Lemma 13. *The adaptations and assertions in a simulator cycle induced by D making $\text{P2}(x_2)$ or $\text{P2}^{-1}(y_2)$ never cause abort.*

Proof. Consider $\text{P2}^{-1}(y_2)$ first, and assume $y_2 \notin P_2^{-1}$. In this case, there would be an assertion, which fails if there exist more than one k such that $y_2 \oplus k \in P_3$. But this is not possible, as otherwise the more than one involved 3-queries would form 33-shoots, and it has to fall into either of the two cases: (i) the shoots are in *Border* and thus in *DUShoots* by Lemma 11, and G_2 should have aborted in the earlier call to $\text{CHECKDUNAWARE}(x_2, X2)$; (ii) the shoots are not in *Border* and thus $y_2 \in P_2^{-1}$ by Inv8 and Inv7. Thus this assertion never causes abort.

We note that adaptations and assertions occur in the rest part of this cycle only if there exists exactly one (K, k) such that $x_3 = y_2 \oplus k \in P_3$. Let the involved 3-query be $(3, x_3, y_3)$, then we distinguish two possibilities:

(i) first, $y_3 \notin ETable[K]^{-1}$. Then the call to $\text{EIN}^{-1}(K, y_3)$ would lead to creating a new E-query $(K, x_1, y_3, \leftarrow)$. By Inv4 and Inv5, right after this point, it holds: (i) $\forall K' \neq K, x_1 \notin ETable[K']$; (ii) $x_1 \notin P_1$. By this, the subsequent call to $\text{P1IN}(x_1)$ would lead to a process similar to that analyzed in Lemma 12, and thus the adaptations and assertions would not cause abort;

(ii) second, $y_3 \in ETable[K]^{-1}$. Let the involved E-query be (K, x_1, y_3). It necessarily holds $x_1 \notin P_1$, as otherwise $(1, K, x_1) \in Completed$ by Inv6 and $y_2 \in P_2^{-1}$ before the cycle. Thus in this case, the subsequent call to $\text{P1IN}(x_1)$ would also lead to a process similar to that analyzed in Lemma 12 (thus no abortion).

This finishes the analysis for $\text{P2}(x_2)$. For $\text{P2}^{-1}(y_2)$ it's similar by symmetry. □

Lemma 14. *The adaptations and assertions in a simulator cycle induced by D making $\text{H}(K)$ never cause abort.*

Proof. Assuming $K \notin HTable$. G_2 first gets $k \leftarrow \mathbf{R}.\text{H}(K)$, and then checks the "goodness" of k. If G_2 does not abort in this phase, then it holds (can be seen from these conditions)

$$\forall (1, x_1, y_1) \in Queries, y_1 \oplus k \notin P_2 \text{ and } \forall k' \in \mathcal{Z} \setminus \{k\}, y_1 \oplus k \oplus k' \notin P_1^{-1}, \tag{2}$$

and

$$\forall (3, x_3, y_3) \in Queries, x_3 \oplus k \notin P_2^{-1} \text{ and } \forall k' \in \mathcal{Z} \setminus \{k\}, x_3 \oplus k \oplus k' \notin P_3. \tag{3}$$

By construction, G_2 then makes a call to $\text{CHECK}(x_1, y_1, K)$ for each query-pair $(1, x_1, y_1)$ and $(3, x_3, y_3)$.[14] If this CHECK-call returns **true**, then it indicates $(K, x_1, y_3, edir, enum) \in EQueries$. G_2 then makes a call to $\text{ADAPT}(2, x_2, y_2, edir, enum)$ for $x_2 = y_1 \oplus k$ and $y_2 = x_3 \oplus k$ to complete the chain corresponding to (K, x_1, y_3). According to (2) and (3), right after k is got from $\mathbf{R}$, it holds $x_2 \notin P_2$ and $y_2 \notin P_2^{-1}$. We note that the query-pairs processed in this cycle have to be distinct: for example, for $y_1' \neq y_1$, $\text{CHECK}(x_1, y_1, K)$ and $\text{CHECK}(x_1, y_1', K)$

[14] It must be $(1, K, x_1) \notin Completed$, otherwise $K \in HTable$ by Lemma 1.

cannot both return **true**. Thus $x_2 \notin P_2$ and $y_2 \notin P_2^{-1}$ keep holding till the ADAPT-call, and thus the call does not cause abortion. As a result, the subsequent UPDATECOMPLETED-call does not cause abortion either. Furthermore, the subsequent assertion never fails by (2) and (3).

After the above chain-completing process, G_2 would check an assertion, which essentially states that for each AD-2-query $(2, x_2, y_2, \perp)$ and each (K, k), the edge $(x_2 \oplus k, y_2 \oplus k)$ is in $AD2Edges$. This assertion never fails because:

(i) When $(2, x_2, y_2)$ is created, by the code of ADAPT, each H-query $(K, k) \in HQueries$ would lead to G_2 adding an edge $(x_2 \oplus k, y_2 \oplus k)$ to $AD2Edges$. Moreover, since G_2 called ADAPT, G_2 is necessarily completing a chain; this implies $|HQueries| \geq 1$, and thus the ADAPT-call could add (at least one) AD-2-edges to $AD2Edges$ successfully;

(ii) Since $(2, x_2, y_2)$ is created, each newly created H-query (K, k) would lead to G_2 adding $(x_2 \oplus k, y_2 \oplus k)$ to $AD2Edges$ in the call to $\mathrm{H}(K)$.

The above complete the analysis. ◻

8.2 Long Simulator Cycles Can be Correctly Handled

This subsection devotes to prove the non-abortion of adaptations and assertions in long simulator cycles. By the pseudocode, during such cycles, creations of new queries, adaptations, and assertions would emerge in calls to RANDASSIGN, COLLECTTP, PROCESS11SHOOT, PROCESS33SHOOT, PROCESS21TP and PROCESS23TP. Another call that would emerge is EMPTYQUEUE, but such calls would not have any "interesting" effects.

The whole analysis would undoubtedly be long and depressing. To remedy this situation, we divide the analysis into several parts, summarized by several propositions:

(1) First, Proposition 17 claims that COLLECTTP never aborts;
(2) Second, Propositions 18 and 19 analyzed the maximal effects that can be brought in by G_2 processing MidTPs and shoots;
(3) Third, based on these mentioned effects, we define PROCESSSHOOT-calls that satisfy certain constraints as *safe* in definition 4, and then shows all such calls are indeed safe in Proposition 21;
(4) Forth, Proposition 22 shows the non-abortion of assertions and adaptations in calls to PROCESS21TP and PROCESS23TP, while Proposition 23 establishes similar claims for PROCESSSHOOT-calls;
(5) Finally, Lemma 15 gathering the conclusions above and complete the proof.

We first give an observation: in a PROCESSSHOOT-call, the 1- and 3-queries "anchored" at the old E-chain either have $qnum$ less than the current $cycleStartNum$ value, or head towards B_2.

Proposition 16. *Assume that D queries $\mathrm{P1}^{-1}(z) \to z'$ or $\mathrm{P3}(z) \to z'$ which triggers a long simulator cycle. Consider the old E-chain of any* PROCESSSHOOT*-call in this cycle $x'_{1,1} - y'_{1,1} - x'_{2,1} - y'_{2,1} - \ldots$. For any 1-query $(1, x'_1, y'_1, d'_1, n'_1)$ (3-query $(3, x'_3, y'_3, d'_3, n'_3)$, resp.) such that $x'_1 \in EB(x'_{1,1})$ ($y'_3 \in EB(x'_{1,1})$, resp.), it holds either $n'_1 < cycleStartNum$ ($n'_3 < cycleStartNum$, resp.), or $d'_1 = \rightarrow$ ($d'_3 = \leftarrow$, resp.).*

Proof. Consider $(1, x'_1, y'_1, d'_1, n'_1)$ first. Towards a contradiction, assuming both $n'_1 \geq cycleStartNum$ and $d'_1 \neq \rightarrow$. Then $x'_1 \in EB(z')$: if $d'_1 = \perp$ then it follows from Lemma 4 (ii), whereas if $d'_1 = \leftarrow$ then it has to be $x'_1 = z'$. By Proposition 2, x'_1 cannot be reached from any vertex $z^* \notin EB(z')$; thus it necessarily be that G_2 detects a shoot formed by two 1- or 3-queries that are both "anchored" at $EB(z')$ (it might be that $(1, x'_1, y'_1)$ itself is involved in this shoot), and when processing this shoot, it takes a path in $EB(z')$ as the old E-chain, so that $(1, x'_1, y'_1)$ could be adjacent to the old E-chain of some PROCESSSHOOT-call. For example, it might be that G_2 detects a shoot formed by two AD-1-queries. However, this is notk possible, as otherwise G_2 would have aborted in a previous call to COLLECTTP. For $(3, x'_3, y'_3)$ the argument is similar. ◻

Then we claim that COLLECTTP never aborts.

Proposition 17. *Calls to* COLLECTTP *never cause abort.*

Proof. Assume that the current long cycle is induced by D querying $\mathrm{P1}^{-1}(z) \to z'$ or $\mathrm{P3}(z) \to z'$. As captured by the assertions, COLLECTTP-calls may abort in two cases: first, values in $Border$ are involved in newly detected shoots; second, unexpectedly detected shoots or TPs appear. The former type is clearly not possible:

the values involved in detected shoots necessarily have their DAWARENESS function values equal 1 (Proposition 8), while the values in *Border* always have their DAWARENESS function values equal 0 (Lemma 11).

We then focus on the latter type of abortion. First, consider the two assertions around newly-detected MidTPs, which require G_2 to abort if a newly-created 1-/3-query form a new MidTP with a 2-query newly created in this cycle (i.e. num_2 is larger than the current $cycleStartNum$ value). In this cycle, new 2-queries can only be created in calls to PROCESS11SHOOT and PROCESS33SHOOT; such new 2-queries are necessarily adapted ones, and thus can not form new MidTPs by Proposition 4. Thus these two assertions never fail.

Second, consider the two assertions around newly-detected shoots, which require G_2 to abort if a newly-created 1-/3-query form a new shoot with a 1-/3-query newly created in this cycle (i.e. $num' \geq cycleStartNum$). First, note that the COLLECTTP-call for $(1, z', z, \leftarrow)$ or $(3, z, z', \rightarrow)$ would not abort at this stage, because when this call is made, $(1, z', z)$ or $(3, z, z')$ is the *only* query with $num \geq cycleStartNum$. Thus we focus on newly created AD-1-queries (which is wlog). For any such 1-query $(1, x_1, y_1, \perp)$, it falls into two possibilities:

- it is created by PROCESS23TP, i.e. there exist $(2, x_2, y_2, n_2)$ and (K, k) s.t. $n_2 < cycleStartNum$ and $x_2 \oplus k = y_1$;
- it is created by PROCESSSHOOT, i.e. when $(1, x_1, y_1, \perp)$ is created, there exist $(1, x_1', y_1', d_1', n_1')$, (K, k), and (K', k') s.t. $y_1' \oplus k \oplus k' = y_1$ and $(1, x_1, \{K, K'\}) \notin ProcessedShoots$.

Then, note that in this cycle, new 1-queries may have directions either $\rightarrow$, or $\leftarrow$, or $\perp$. The first type clearly cannot form new (unprocessed) 11-shoot with $(1, x_1, y_1, \perp)$. For this, assume otherwise, i.e. a new query $(1, x_1^\circ, y_1^\circ, \rightarrow, n_1^\circ)$ satisfy $y_1^\circ = y_1 \oplus k'' \oplus k'''$ and $(1, x_1, \{K'', K'''\}) \notin ProcessedShoots$ for some $k'', k''' \in \mathcal{Z}$. Then:

- If there exist $(2, x_2, y_2, n_2)$ and (K, k) s.t. $n_2 < cycleStartNum$ and $x_2 \oplus k = y_1$, then $n_2 < cycleStartNum < n_1^\circ$ which contradicts Inv2;
- If there exist $(1, x_1', y_1', d_1', n_1')$, (K, k), and (K', k') s.t. $y_1' \oplus k \oplus k' = y_1$ when $(1, x_1, y_1, \perp)$ is created, then $(1, x_1', y_1')$ and $(1, x_1^\circ, y_1^\circ)$ cannot be the same query, otherwise either $\{K'', K'''\} = \{K, K'\}$ and $(1, x_1, \{K, K'\}) \in ProcessedShoots$, or $k \oplus k' \oplus k'' \oplus k''' = 0$ contradicts Inv1.[15] Thus:
  - if $n_1' \leq cycleStartNum < n_1^\circ$ then it contradicts Inv3;
  - if $n_1' > cycleStartNum$ then $d_1' =\rightarrow$ by Proposition 16 and it necessarily contradicts Inv3.

On the other hand, the other two types of new 1-queries cannot form new 11-shoot with $(1, x_1, y_1, \perp)$ either. For this, assume otherwise, i.e. a new query $(1, x_1^\circ, y_1^\circ, d_1^\circ, n_1^\circ)$ satisfies $y_1^\circ = y_1 \oplus k'' \oplus k'''$ for some $k'', k''' \in \mathcal{Z}$. Then according to Lemma 4 (ii), it holds $y_1 \in B_2(z')$ and $y_1^\circ \in B_2(z')$ (when $n_1^\circ = cycleStartNum$ we have $y_1^\circ = z'$, thus $y_1^\circ \in B_2(z')$ also holds). This implies that there exists a "pseudo-cycle" in B_2: $y_1 - \ldots z' - \ldots - y_1^\circ \underline{\oplus k'' \oplus k'''} (y_1)$, and the impossibility is established similarly to Proposition 5. The above establishes the claim. □

Effects of G_2 Processing Shoots and MidTPs. The next proposition describes the influences of non-aborting PROCESS21TP- and PROCESS23TP-calls on the trees anchored at the arguments.

Proposition 18. *A non-aborting call to* PROCESS21TP$(x_1^\circ, y_1^\circ, K)$ *(*PROCESS23TP$(x_3^\circ, y_3^\circ, K)$*, resp.) has at most two effects on $EB(x_1^\circ)$ and $B_2(y_1^\circ)$ ($EB(x_3^\circ)$ and $B_2(y_3^\circ)$, resp.) as follows:*

(i) Attaching a new edge labeled K to x_1° (y_3°, resp.);
(ii) Making y_1° (x_3°, resp.) pebbled.

Proof. By the code, a call to PROCESS21TP$(x_1^\circ, y_1^\circ, K)$ would consist of two "relevant" operations. First, it calls EIN(K°, x_1°), which has two possibilities:

- if $x_1^\circ \in ETable[K]$ before this EIN-call, then no new E-query would be created;
- if $x_1^\circ \notin ETable[K]$ before this EIN-call, then a new E-query $(K, x_1^\circ, y_3^\circ, \rightarrow)$ would be created, with $y_3^\circ = \mathbf{E}.\mathrm{E}(K, x_1^\circ)$. By Inv4, right after this point, y_3° is not adjacent to any E-query except for $(K, x_1^\circ, y_3^\circ)$. Thus exactly one edge is attached to x_1°.

The above matches (i). Second, it calls ADAPT$(3, x_3^\circ, y_3^\circ)$, which will make x_3° pebbled (if it has not been pebbled yet). This matches (ii). The case of PROCESS23TP is similar by symmetry. □

The next proposition considers the influences of non-aborting PROCESSSHOOT-calls.

[15] Recalling from subsection 4.5: given Inv1, *a fixed pair of 1-queries can form at most one 11-shoot.*

Proposition 19. *After a non-aborting call to* PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_1, K_2)$*, it holds:*

(i) for any $1 \le l \le 2t$*,* $xebval_l(K_1, K_2, x_1^\circ) \neq \perp$*,* $xebval_l(K_2, K_1, x_1^\circ) \neq \perp$*, and* $xebval_{2t+1}(K_1, K_2, x_1^\circ) = xebval_{2t+1}(K_2, K_1, x_1^\circ) = \perp$*;*

(ii) Among the nodes of the form $yb2val_l(K_1, K_2, y_1^\circ)$ *and* $yb2val_l(K_1, K_2, y_1^\circ)$*, at most* $4t$ *nodes are newly-pebbled by this call, i.e. those with* $l = 1, 2, \ldots, 2t$*.*

Symmetrically, after a non-aborting call to PROCESS33SHOOT$(x_3^\circ, y_3^\circ, K_1, K_2)$*, it holds:*

(i) for any $1 \le l \le 2t$*,* $yebval_l(K_1, K_2, y_3^\circ) \neq \perp$*,* $yebval_l(K_2, K_1, y_3^\circ) \neq \perp$*, and* $yebval_{2t+1}(K_1, K_2, y_3^\circ) = yebval_{2t+1}(K_2, K_1, y_3^\circ) = \perp$*;*

(ii) Among the nodes of the form $xb2val_l(K_1, K_2, x_3^\circ)$ *and* $xb2val_l(K_1, K_2, x_3^\circ)$*, at most* $4t$ *nodes are newly-pebbled by this call, i.e. those with* $l = 1, 2, \ldots, 2t$*.*

Proof. Wlog we focus on PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_1, K_2)$, and let $y_1^{\circ\circ} = y_1^\circ \oplus k_1 \oplus k_2$ and $x_1^{\circ\circ} = P_1^{-1}(y_1^{\circ\circ})$. As mentioned before, the subsequent process is divided into four phases: the *Make-E-Chain-Phase*, the *Shoot-Growing-Phase*, the *Fill-in-Rung-Phase*, and the *Shoot-Completing-Phase*. To avoid taking us afield, we eschew the concrete statements in favor of informal descriptions.

In the *Make-E-Chain-Phase*, G_2 would take $x_1^{\circ\circ}$ and x_1° as the "starting points" and make $2 \cdot 4t$ queries to EIN/EIN^{-1}. To save page, we follow the notations used in Lemma 3. This would result in two E-chains of length $2t$ that are adjacent to x_1°, thus $xebval_l(K_1, K_2, x_1^\circ) \neq \perp$ and $xebval_l(K_2, K_1, x_1^\circ) \neq \perp$ hold for any $1 \le l \le 2t$. On the other hand, when G_2 reaches $y_{1,1}^{\circ\circ}$ when it is evaluating along the old E-chain, if $y_{1,1}^{\circ\circ} \in ETable[K_2]^{-1}$, then $(1, \{(x_{1,1}^\circ, y_{1,1}^\circ), (x_{1,1}^{\circ\circ}, y_{1,1}^{\circ\circ})\})$ (with $x_{1,1}^\circ$ and $x_{1,1}^{\circ\circ}$, the two values that are supposed to be in *Border*) would not be added to *DUShoots*, contradicting Lemma 11. Thus it necessarily holds $y_{1,1}^{\circ\circ} \notin ETable[K_2]^{-1}$, and the two E-queries $(K_2, x_{1,1}^{\circ\circ}, y_{1,1}^{\circ\circ}, \leftarrow)$ and $(K_2, x_{1,1}^{\circ\circ}, y_{1,1}^{\circ\circ}, \leftarrow)$ would be new, and $xebval_{2t+1}(K_1, K_2, x_1^\circ) = xebval_{2t+1}(K_2, K_1, x_1^\circ) = \perp$ by Inv4 and (i) is established.

On the other hand, the AD-2-queries created in the *Fill-in-Rung-Phase* would make $yb2val_l(K_1, K_2, y_1^\circ)$ and $yb2val_l(K_1, K_2, y_1^\circ)$ change to non-empty for every $1 \le l \le 2t$ (if they were $\perp$).[16] None of these $4t$ nodes could be pebbled by the 1- and 3-queries created in *Shoot-Growing-Phase*, since these queries head towards B_2.[17] Therefore, the only mechanism that pebbles them is the *Shoot-Completing-Phase*, which indeed pebbles exactly all of them. These establishes (ii). □

Safe Calls to ProcessShoot. PROCESSSHOOT-calls that meet certain constraints would be called *safe* (this terminology is borrowed from [LS13], though the details deviate a lot).

Definition 4. *A call to* PROCESS11SHOOT(x_1', y_1', K_1, K_2) *is* safe *if for any* $l \ge 2$*, it holds* $xebval_l(K_1, K_2, x_1') = xebval_l(K_2, K_1, x_1') = \perp$ *right before the call is made;*[18] *symmetrically, a call to* PROCESS33SHOOT(x_3', y_3', K_1, K_2) *is* safe *if for any* $l \ge 2$*, it holds* $yebval_l(K_1, K_2, y_3') = yebval_l(K_2, K_1, y_3') = \perp$ *right before the call.*

Safe calls to PROCESSSHOOT are easier to analyze. Indeed, we will show that all of them are safe. We first point out a helper property brought in by the design of PROCESSSHOOT procedures: shoots are processed in a strict order.

Proposition 20. *Assume that two shoots* $(i, z^\circ, \{K_1, K_2\})$ *and* $(j, z^{\circ\circ}, \{K_1', K_2'\})$ *are popped and processed in the same (long) simulator cycle due to* D *querying* $\mathrm{P1}^{-1}(z) \to z'$ *or* $\mathrm{P3}(z) \to z'$*, with* $(i, z^\circ, \{K_1, K_2\})$ *being popped earlier. Then* z° *cannot lie beneath* $z^{\circ\circ}$ *in* $EB(z')$*.*

Proof. This can been seen from the order of adaptations in the *Shoot-Completing-Phase* of PROCESSSHOOT procedure. Briefly speaking, if z° lies beneath $z^{\circ\circ}$, then AD-1- and AD-3-queries are necessarily first attached to $z^{\circ\circ}$ and then to z°, thus the shoots rooted at $z^{\circ\circ}$ are necessarily closer to the front of *ShootQueue* than the shoots rooted at z° – and would be popped earlier. □

Then the main claim:

16 One could see Fig. 3 (bottom right) for an illustration. However, to give a formal argument seems intricate.

17 Note that in this phase, G_2 processing 13- and 31-TPs may attach new AD-2-edges to $B_2(y_1^\circ)$, but would not create new 1- and 3-queries and thus not pebbling any nodes.

18 This implies: (i) $x_1' \notin ETable[K_1]$ or $ETable[K_1](x_1') \notin ETable[K_2]^{-1}$, and (ii) $x_1' \notin ETable[K_2]$ or $ETable[K_2](x_1') \notin ETable[K_1]^{-1}$.

Proposition 21. *All calls to* ProcessShoot *are safe.*

Proof. Wlog consider a call to Process11Shoot(x'_1, y'_1, K_1, K_2), and assume that its simulator cycle is induced by D querying $\mathrm{P1}^{-1}(z) \to z'$ or $\mathrm{P3}(z) \to z'$. Then by Lemma 4 (ii) it holds $x'_1 \in EB(z')$ before Process11Shoot(x'_1, y'_1, K_1, K_2).

We first argue that right after x'_1 becomes a node of $EB(z')$ (i.e. $x'_1 \in EB(z')$ holds), it holds

$$\underline{xebval_2(K_1, K_2, x'_1) = xebval_2(K_2, K_1, x'_1) = \bot}.$$

This is clear when $x'_1 = z'$, since right after $(1, z', z, \leftarrow)$ or $(3, z, z', \rightarrow)$ is created, z' is not adjacent to any edge in EB. Otherwise, by Lemma 4 (i), the path between z' and x'_1 is directed from z' to x'_1. This implies the existence of an E-query $(K^*, x'_1, y^*_3, \leftarrow)$ for some y^*_3. By Inv4, among all the queries that will be adjacent to x'_1, this query has the least $qnum$ value; thus right after (K^*, x'_1, y^*_3) is created, by Inv4 it holds

$$\forall K \neq K^*, x'_1 \notin ETable[K]. \tag{4}$$

We now argue $K^* \notin \{K_1, K_2\}$ – indeed, we are trying to show if $K^* = K_1$ or K_2 then G_2 popping $(1, x'_1, \{K_1, K_2\})$ would not lead to calling Process11Shoot. Assume otherwise, and wlog assume $K^* = K_1$. Then by Proposition 6, (K_1, x'_1, y^*_3) cannot have been created during G_2 processing a MidTP. Thus assuming G_2 was processing a shoot of the form $(i, z^\circ, \{K_1, K_3\})$ with $K_3 \neq K_2$.[19] According to the code and the assumptions, the following hold right before G_2 creates $(1, x'_1, y'_1, \bot)$, cf. Fig. 8 (left):

(i) there exists a 1-query $(1, x''_1, y''_1, n''_1, d''_1)$ with $y''_1 = y'_1 \oplus k_1 \oplus k_3$ (this query is involved in G_2 processing $(i, z^\circ, \{K_1, K_3\})$, and G_2 computes y'_1 as $y'_1 \leftarrow y''_1 \oplus k_1 \oplus k_3$, cf. the code of ProcessShoot procedures);
(ii) there exists a 1-query $(1, x'''_1, y'''_1, n'''_1, d'''_1)$ with $y'''_1 = y'_1 \oplus k_1 \oplus k_2$ (so that G_2 could detect $(1, x'_1, \{K_1, K_2\})$ after creating $(1, x'_1, y'_1, \bot)$).

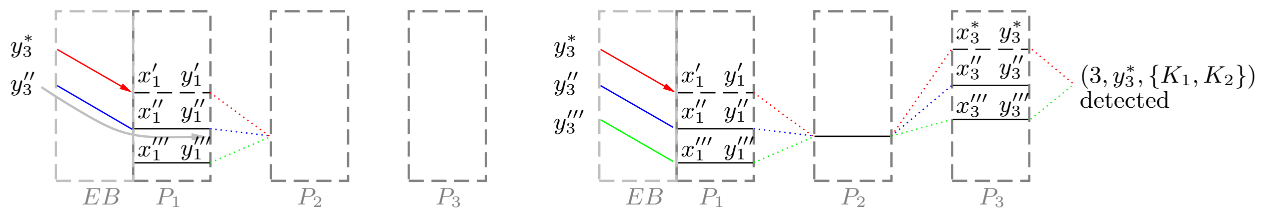


Fig. 8. For Proposition 21: the lines in red, green, and blue indicate E-queries labeled K_1, K_2, and K_3 respectively, while the colored dotted lines indicate the connection under the corresponding round-keys. (left) the two 1-queries supposed to exist. The arrowed silver line indicates the direction of G_2's evaluation during processing $(i, z^\circ, \{K_1, K_3\})$; (right) the 1-queries imply completed paths as well as G_2 detecting $(3, y^*_3, \{K_1, K_2\})$ earlier.

We proceed to argue $n''_1, n'''_1 < cycleStartNum$. For this, we note:

(i) If $n''_1 \geq cycleStartNum$ and $n''_1 > n'''_1$, then $d''_1 = \rightarrow$ by Proposition 16. But this contradicts Inv3;
(ii) If $n'''_1 \geq cycleStartNum$ and $n'''_1 > n''_1$, then $d'''_1 \in \{\leftarrow, \bot\}$ by Inv2. Thus right before $(1, x'_1, y'_1, \bot)$ is created, it holds:
  (a) $y'_1 \in B_2(z)$ (by Lemma 4 (ii));
  (b) $y'''_1 \in B_2(z)$ (if $d'''_1 = \bot$ then this follows from Lemma 4 (ii), otherwise it holds $y'''_1 = z$);
  (c) $y'_1 \oplus y'''_1 = k_1 \oplus k_2 \in 2\mathcal{Z}$.
  Thus by an argument similar to Proposition 17, we could show that at some point before $(1, x'_1, y'_1, \bot)$ is created, G_2 should have aborted. As a consequence, the purported call to Process11Shoot(x'_1, y'_1, K_1, K_2) should not have been possible, a contradiction.

It's not hard to see that the above have excluded all the possibilities of n''_1 or $n'''_1 \geq cycleStartNum$. Thus $n''_1, n'''_1 < cycleStartNum$. By Proposition 15 we got $x''_1 \notin Border$, thus $(1, K_3, x''_1), (1, K_2, x'''_1) \in Completed$ before this cycle by Inv8 and Inv7. This implies the existence of four queries (K_3, x''_1, y''_3), $(3, x''_3, y''_3)$, (K_2, x'''_1, y'''_3),

[19] If $K_3 = K_2$ then there would not be any sub-call to Process11Shoot(x'_1, y'_1, K_1, K_2) because $(1, x'_1, \{K_1, K_2\}) \in ProcessedShoot$, cf. the code of CollectTP procedures.

and $(3, x_3''', y_3''')$ before this cycle, with $x_3''\oplus x_3''' = k_2\oplus k_3$. Consider the point right before G_2 creating $(1, x_1', y_1', \perp)$. By the pseudocode, it can be seen that G_2 must have created the 3-query $(3, x_3^*, y_3^*, d_3^*)$ before this point (d_3^* may be $\rightarrow$ or $\perp$, but this does not matter), and as G_2 detects $x_3''' = x_3^*\oplus k_1\oplus k_2$, a 33-shoot $(3, y_3^*, \{K_1, K_2\})$ must have been pushed into *ShootQueue* before $(1, x_1', \{K_1, K_2\})$ is pushed, cf. Fig. 8 (right). This shoot $(3, y_3^*, \{K_1, K_2\})$ would be popped earlier than $(1, x_1', \{K_1, K_2\})$, leading to a call to $\textsc{Process33Shoot}(P_3^{-1}(y_3^*), y_3^*, K_1, K_2)$, after which $(1, x_1', \{K_1, K_2\})$ would be in *ProcessedShoot* (as they are obviously equivalent), and thus the purported call to $\textsc{Process11Shoot}(x_1', y_1', K_1, K_2)$ would not have happened when $(1, x_1', \{K_1, K_2\})$ is (later) popped. This contradicts the assumption.

By the above, it holds $K^* \notin \{K_1, K_2\}$, so that right after $x_1' \in EB(z')$ holds, we have $x_1' \notin ETable[K_1]$ and $x_1' \notin ETable[K_2]$, and thus $xebval_2(K_1, K_2, x_1') = xebval_2(K_2, K_1, x_1') = \perp$.

We then argue that $xebval_2(K_1, K_2, x_1') = xebval_2(K_2, K_1, x_1') = \perp$ is kept till $(1, x_1', \{K_1, K_2\})$ is popped and processed. We first note that if at some point, G_2 detects a 23-TP $(3, y_{3,1}, K_2)$ for $y_{3,1} = ETable[K_1](x_1')$, and this 23-TP is popped (and processed) before $\textsc{Process11Shoot}(x_1', y_1', K_1, K_2)$, then $xebval_2(K_1, K_2, x_1')$ would be changed to non-empty. However, the possibility is ruined out by Proposition 6. Similarly, G_2 detecting a 23-TP $(3, ETable[K_2](x_1'), K_1)$ is not possible either. According to Propositions 18, these are the only cases that earlier-processed MidTPs can affect $xebval_2(K_1, K_2, x_1')$ and $xebval_2(K_2, K_1, x_1')$. Thus MidTPs are excluded.

We then show that the two values cannot be affected by earlier-processed shoots either. Briefly speaking, this relies on the order of adaptations in the *Shoot-Completing-Phase*. More clearly, since $K^* \notin \{K_1, K_2\}$, by Propositions 20 and 19, all the shoots that simultaneously meet the following constraints are necessarily of the form $(3, y_3^*, \{K^*, K_1\})$, $(3, y_3^*, \{K^*, K_2\})$, or $(1, x_1', \{K_1, K_3\})$ with $K_3 \neq K_1, K_2$, or $(1, x_1', \{K_2, K_4\})$ with $K_4 \neq K_1, K_2$ (note we assumed $(K^*, x_1', y_3^*, \leftarrow)$):

(i) they are popped earlier than $(1, x_1', \{K_1, K_2\})$;
(ii) they are able to attach edges to x_1'.[20]

By an inspection of these cases and Proposition 19, one could see that none of them is able to change $xebval_2(K_1, K_2, x_1')$ and $xebval_2(K_2, K_1, x_1')$ to non-empty.

By the above, $xebval_2(K_2, K_1, x_1')$ remain $\perp$ till the call to $\textsc{Process11Shoot}(x_1', y_1', K_1, K_2)$. A symmetrical argument shows that $xebval_2(K_2, K_1, x_1')$ is also kept $\perp$. Thus the claim. ◻

After all the preparations above, we are now able to present the non-abortion arguments for G_2 processing MidTPs and shoots. We first consider MidTPs.

MidTPs Can be Handled. Formally stated as the following proposition.

Proposition 22. *Adaptations and assertions in calls to* $\textsc{Process21TP}$ *and* $\textsc{Process23TP}$ *never lead to abortion.*

Proof. Consider such a call to $\textsc{Process21TP}(x_1^\circ, y_1^\circ, K^\circ)$, and assume that its simulator cycle is induced by D querying $\mathrm{P1}^{-1}(z) \rightarrow z'$ or $\mathrm{P3}(z) \rightarrow z'$.

First, we argue that right before this call is made, it holds:

(i) $x_1^\circ \notin ETable[K^\circ]$;
(ii) the vertex $x_3^\circ = k^\circ \oplus P_2(k^\circ \oplus y_1^\circ)$ is not pebbled.

For Claim (i): We first argue that right after $x_1^\circ \in EB(z')$ holds, it holds $x_1^\circ \notin ETable[K^\circ]$. This is clear when $x_1^\circ = z'$, since right after $(1, z', z, \leftarrow)$ or $(3, z, z', \rightarrow)$ is created by $\textsc{RandAssign}$, z' is not adjacent to any edge in EB. Otherwise, by Lemma 4 (i), the path is directed from z' to x_1°. This implies the existence of an E-query $(K^*, x_1^\circ, y_3^*, \leftarrow)$ for some y_3^*. By Inv4, among all the queries that will be adjacent to x_1°, this query has the least *qnum* value; thus right after $(K^*, x_1^\circ, y_3^*, \leftarrow)$ is created (and $x_1^\circ \in EB(z')$ holds), by Inv4 it holds

$$\forall K \neq K^*, x_1^\circ \notin ETable[K]. \tag{5}$$

[20] Shoots of the form e.g. $(1, x_1^*, \{K^*, K_1\})$ with $x_1^* = ETable[K^*](y_3^*)$ also meet the constraints, but note that $(1, x_1^*, \{K^*, K_1\}) \equiv (3, y_3^*, \{K^*, K_1\})$.

Moreover, $K^* \neq K^\circ$, as otherwise $(K^\circ, x_1^\circ, y_3^*)$ is dead after the call which creates it due to Lemma 3, which implies $(1, K^\circ, x_1^\circ) \in \mathit{Completed}$ and the purported call to PROCESS21TP$(x_1^\circ, y_1^\circ, K^\circ)$ should not have happened (cf. the code of EMPTYQUEUE). Hence claim (i) holds right after $x_1^\circ \in EB(z')$ holds.

We then argue that from this point till the call to PROCESS21TP$(x_1^\circ, y_1^\circ, K^\circ)$ is made, $x_1^\circ \in \mathit{ETable}[K^\circ]$ is never possible. Assume otherwise, then there must be a detected shoot that is processed in this period, and G_2 processing this shoot leads to calling EIN(K°, x_1°). We will show this is impossible: briefly speaking, if such a shoot exists, then there must be some additional queries around x_1° that should have led to G_2 detecting a shoot of the form $(1, x_1^\circ, \{K^\circ, K'\})$ rather the purported 21-TP $(1, K^\circ, x_1^\circ)$.

In detail, according to Propositions 18 and 19, it can be seen that the following three cases would lead to G_2 calling EIN(K°, x_1°):

Case 1.1: At some point, G_2 detects a 11-shoot of the form $(1, x_1^\circ, \{K^\circ, K'\})$ for some $K' \neq K^\circ$, and this shoot is popped (and processed) before $(1, K^\circ, x_1^\circ)$ is popped. The possibility of this case is immediately ruined out by Proposition 6.

Case 1.2: G_2 detects a 33-shoot of the form $(3, y_{3,1}, \{K^\circ, K_1\})$ for some $K_1 \neq K^\circ$ and $y_{3,1} = \mathit{ETable}[K_1](x_1^\circ)$, and this shoot is popped before $(1, K^\circ, x_1^\circ)$ is popped.

By Proposition 6, it is not possible that the E-query $(K_1, x_1^\circ, y_{3,1}, \rightarrow)$ is created when G_2 is processing a MidTP. Assume that $(K_1, x_1^\circ, y_{3,1})$ is created when G_2 is processing a shoot of the form $(i, z^\circ, \{K_1, K_2\})$. This implies the existence of a 1-query $(1, x_1^{\circ\circ}, y_1^{\circ\circ}, d_1^{\circ\circ}, n_1^{\circ\circ})$ with $y_1^{\circ\circ} = y_1^\circ \oplus k_1 \oplus k_2$. Furthermore, when G_2 is processing the purported shoot $(i, z^\circ, \{K_1, K_2\})$, there necessarily exists a point such that if we let $y_{3,1} = \mathit{ETable}[K_1](x_1^\circ)$, $y_3^{\circ\circ} = \mathit{ETable}[K_2](x_1^{\circ\circ})$, and let the two involved 3-queries be $(3, x_3^{\circ\circ}, y_3^{\circ\circ}, d_3^{\circ\circ}, n_3^{\circ\circ})$ and $(3, x_{3,1}, y_{3,1}, \perp, n_{3,1})$, then there exists a 3-query $(3, x'_{3,1}, y'_{3,1}, d'_{3,1}, n'_{3,1})$ with $x'_{3,1} = x_{3,1} \oplus k_1 \oplus k^\circ$ (so that G_2 would detect the purported 33-shoot $(3, y_{3,1}, \{K_1, K^\circ\})$ after creating $(3, x_{3,1}, y_{3,1})$), cf. Fig. 9 (left).

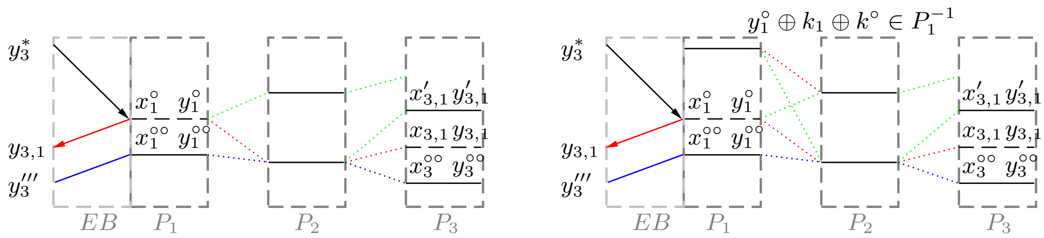


Fig. 9. For Proposition 22: lines in green, red, and blue indicate E-queries/relations keyed (K°, k°), (K_1, k_1), and (K_2, k_2) respectively. (left) the involved 1- and 3-queries; (right) the implication.

It necessarily be $n'_{3,1} < \mathit{cycleStartNum}$, as otherwise G_2 should have aborted in COLLECTTP$(3, x_{3,1}, y_{3,1})$ when detecting $x_{3,1} = x'_{3,1} \oplus k_1 \oplus k^\circ$. On the other hand, it also holds $n_3^{\circ\circ} < \mathit{cycleStartNum}$, as otherwise $d_3^{\circ\circ} = \leftarrow$ by Proposition 16 and $x'_{3,1} = x_{3,1} \oplus k_1 \oplus k^\circ$ is not possible by Inv2. $n_3^{\circ\circ} < \mathit{cycleStartNum}$ also implies that $y_3^{\circ\circ}$ cannot be the "endpoints" of the old E-chain corresponding to $(i, z^\circ, \{K_1, K_2\})$; thus $y_3^{\circ\circ} \notin \mathit{Border}$ by Proposition 15. Thus before this cycle, $(3, K^\circ, x'_{3,1}) \in \mathit{Completed}$ by Inv8 and Inv7. This implies $y_1^{\circ\circ} \oplus k_2 \oplus k^\circ = y_1^\circ \oplus k_1 \oplus k^\circ \in P_1^{-1}$, cf. Fig. 9 (right), and thus after creating $(1, x_1^\circ, y_1^\circ)$, G_2 should have detected $(1, x_1^\circ, \{K^\circ, K_1\})$ rather than $(1, K^\circ, x_1^\circ)$.

Case 1.3: For some $K_1 \neq K^\circ$ and $l \geq 2$, G_2 detects a shoot $(i, z^\circ, \{K^\circ, K_1\})$ with $z^\circ = \mathit{xebval}_l(K_1, K^\circ, x_1^\circ)$ or $z^\circ = \mathit{xebval}_l(K^\circ, K_1, x_1^\circ)$, and this shoot is popped before $(1, K^\circ, x_1^\circ)$ is popped. We note that the PROCESSSHOOT-call corresponding to this shoot is *not safe*, cf. definition 4. Thus it contradicts Proposition 21, which states that all PROCESSSHOOT-calls are safe.

The above establish claim (i).

For Claim (ii): We first note that $x_3^\circ \notin P_3$ before this cycle: otherwise $(3, K^\circ, x_3^\circ) \in \mathit{Completed}$ by Inv6, and the purported 21-TP should not have occurred.

We then argue that from this point till the call to PROCESS21TP$(x_1^\circ, y_1^\circ, K^\circ)$ is made, $x_3^\circ \in P_3$ is not possible. According to Propositions 18 and 19, since the effects on the two trees in EB and B_2 are somewhat symmetric, G_2 pebbling x_3° also falls into three cases as follows:

Case 2.1: At some point, for some $K' \neq K^\circ$, G_2 detects a 11-shoot of the form $(1, x_1^\circ, \{K^\circ, K'\})$, which is popped before $(1, K^\circ, x_1^\circ)$ is popped. Immediately ruled out by Proposition 6.

Case 2.2: for some $K_1 \neq K^\circ$ and $y_{3,1} = ETable[K_1](x_1^\circ)$, G_2 detects a 33-shoot of the form $(3, y_{3,1}, \{K^\circ, K_1\})$, which is popped before $(1, K^\circ, x_1^\circ)$ is popped (thus there will be a 2-edge $(y_1^\circ, P_3^{-1}(y_{3,1}), k_1)$, and $x_3^\circ = xb2val_2(k_1, k^\circ, P_3^{-1}(y_{3,1})))$. However, the possibility of such shoots has already been ruined out in the argument for claim (i) (cf. *Case 1.2* above, page 52).

Case 2.3: For some $K_1 \neq K^\circ$ and $l \geq 2$, G_2 detects a shoot of the form $(i, z^\circ, \{K^\circ, K_1\}))$ with $z^\circ = xebval_l(K_1, K^\circ, x_1^\circ)$ or $z^\circ = xebval_l(K^\circ, K_1, x_1^\circ)$, and this shoot is popped before $(1, K^\circ, x_1^\circ)$ is popped (thus it will be $x_3^\circ = xb2val_l(k^\circ, k_1, P_1(z^\circ))$ (when $i = 1$) or $x_3^\circ = xb2val_l(k_1, k^\circ, P_3^{-1}(z^\circ))$ (when $i = 3$)). But the ProcessShoot-call corresponding to this shoot is not safe, contradicting Proposition 21.

Then, based on the above, we argue the adaptations and assertions in this call Process21TP$(x_1^\circ, y_1^\circ, K^\circ)$ do not cause abort. By the code, this call would consist of five "interesting" operations:

(1) Checking an assertion, which fails if $\exists k' \neq k^\circ : y_1^\circ \oplus k^\circ \oplus k' \in P_1^{-1}$;
(2) Calls EIn(K°, x_1°);
(3) Calls CheckDUnaware$(x_3^\circ, X3)$, and checks another assertion;
(4) Calls Adapt$(3, x_3^\circ, y_3^\circ, \perp, \perp)$ and UpdateCompleted$(3, K^\circ, x_3^\circ)$;
(5) Calls CollectTP$(3, x_3^\circ, y_3^\circ)$.

We consider these operations one-by-one. First, the first assertion would not fail, because by the code of CollectTP, if $\exists K' \neq K^\circ : x_1^\circ \oplus k \oplus k' \in P_1^{-1}$ then CollectTP$(1, x_1^\circ, y_1^\circ)$ would ignore the fact $y_1^\circ \oplus k^\circ \in P_2$ and thus $(1, x_1^\circ, K^\circ)$ would not have been in *MidTPQueue*.

Second, as already argued, $x_1^\circ \notin ETable[K^\circ]$ holds right before this call, the EIn-call would lead to creating a new E-query $(K, x_1^\circ, y_3^\circ, \rightarrow)$ (with $y_3^\circ = \mathbf{E}.\mathrm{E}(K^\circ, x_1^\circ)$). By Inv5, right after this point, it holds

$$y_3^\circ \notin P_3^{-1} \text{ and } \forall K' \neq K^\circ, y_3^\circ \notin ETable[K']^{-1}. \tag{6}$$

By this and claim (ii) ($x_3^\circ \notin P_3$ right before this call), the adaptation Adapt$(3, x_3^\circ, y_3^\circ, \perp, \perp)$ would not abort. (Consequently, UpdateCompleted$(3, K^\circ, x_3^\circ)$ would not abort either.)

Finally, the assertion fails if $\exists K \neq K^\circ : ETable[K]^{-1}(y_3^\circ) \in P_1$, which is not possible by (6). ◻

We then focus on shoots.

Shoots Can be Handled. Formally stated as follows:

Proposition 23. *Adaptations and assertions in* ProcessShoot*-calls never lead to abortion.*

Proof. Consider such a call to Process11Shoot$(x_1^\circ, y_1^\circ, K_1, K_2)$, let $y_1^{\circ\circ} = y_1^\circ \oplus k_1 \oplus k_2$ and $x_1^{\circ\circ} = P_1^{-1}(y_1^{\circ\circ})$, and wlog assume that its simulator cycle is induced by D querying $\mathrm{P1}^{-1}(z) \rightarrow z'$ or $\mathrm{P3}(z) \rightarrow z'$. By the code, adaptations and assertions only exist in the *Shoot-Growing-Phase*, the *Fill-in-Rung-Phase*, and the *Shoot-Completing-Phase*. We proceed to analyze one-by-one.

<u>G_2 *Never Aborts During the Shoot-Growing-Phase.*</u> In this phase, G_2 first iterates for all nodes $x'_{1,i}$ in the old E-chain. We proceed to show that G_2 does not abort in this iteration.

By the code, for each $x'_{1,i}$, if $x'_{1,i} \notin P_1$, G_2 would create a new 1-query with direction $\rightarrow$ and (possibly) detect and process several 31-TPs. This (sub-)process is somewhat similar to that analyzed in Lemma 12 (though much more complicated): right after the subsequent RandAssign-call returns $y'_{1,i}$, the following holds by Inv2 and Inv3:

$$\forall z'' \in 5\mathcal{Z}, y'_{1,i} \oplus z'' \notin P_2 \text{ and } \forall z'' \in 6\mathcal{Z}, y'_{1,i} \oplus z'' \notin P_1^{-1}. \tag{7}$$

G_2 then completes a chain for each (K^i, k^i) and $(3, \overline{x_3^i}, \overline{y_3^i}, \overline{d_3^i}, \overline{n_3^i})$ such that the E-query $(K^i, x'_{1,i}, \overline{y_3^i})$ pre-exists. Note that by Proposition 3, this query $(K^i, x'_{1,i}, \overline{y_3^i})$ was necessarily created before this cycle, and DAwareness$(\overline{y_3^i}, Y3) = 1$, thus $\overline{y_3^i} \notin Border$ by Lemma 11.

Consider the chain-completion for $(K^i, x'_{1,i}, \overline{y_3^i})$ and $(3, \overline{x_3^i}, \overline{y_3^i})$. We first prove the non-abortion of the subsequent call to $\textsc{Adapt}(2, \overline{x_2^i}, \overline{y_2^i}, \ldots)$. First, by (7), $\overline{x_2^i} = y'_{1,i} \oplus k^i \notin P_2$ holds right after the $\textsc{RandAssign}$-call. During the period between $\textsc{RandAssign}$ and $\textsc{Adapt}(2, \overline{x_2^i}, \overline{y_2^i}, \ldots)$, there only exist calls to $\textsc{Adapt}(2, \overline{x_2^j}, \overline{y_2^j}, \ldots)$ for $K^j \neq K^i$ which implies $\overline{x_2^j} = y'_{1,i} \oplus k^j \neq \overline{x_2^i}$. Therefore, $\overline{x_2^i} \notin P_2$ cannot be changed by these earlier $\textsc{Adapt}$-calls, and is kept till $\textsc{Adapt}(2, \overline{x_2^j}, \overline{y_2^j}, \ldots)$.

Second, before the simulator cycle, $\overline{y_2^i} = \overline{x_3^i} \oplus k^i \in P_2^{-1}$ is not possible, as otherwise $(3, K^i, \overline{x_3^i}) \in \mathit{Completed}$ by Inv6 and $x'_{1,i} = \mathit{ETable}[K^i]^{-1}(\overline{y_3^i}) \in P_1$, contradicting the earlier assumption $x'_{1,i} \notin P_1$.

Moreover $\overline{y_2^i}$ cannot be added to P_2^{-1} by the earlier $\textsc{Adapt}$-calls. To show this, assume otherwise, then there should have been a call to $\textsc{Adapt}(2, x_2^*, \overline{y_2^i}, \mathit{edir}^*, \mathit{enum}^*)$, as all the 2-queries newly created in this cycle are adapted ones. Assume that this call corresponds to the path ($y_1^* \oplus k^* = x_2^*$ and $x_3^* \oplus k^* = \overline{y_2^i}$)

$$(K^*, k^*), (K^*, x_1^*, y_3^*), (1, x_1^*, y_1^*), (3, x_3^*, y_3^*, d_3^*, n_3^*),$$

then it holds $x_3^* \oplus \overline{x_3^i} = k^* \oplus k^i$. However, this is not possible, as we will exclude each possibility:

(i) If $\overline{n_3^i}, n_3^* < \mathit{cycleStartNum}$, then by $\overline{y_3^i} \notin \mathit{Border}$ (already argued), Inv8, and Inv7 it holds $(3, K^i, \overline{x_3^i}) \in \mathit{Completed}$ and $x'_{1,i} \in P_1$ before the cycle, contradicting the assumption;

(ii) If $\overline{n_3^i}, n_3^* \geq \mathit{cycleStartNum}$, wlog assume $\overline{n_3^i} > n_3^*$, then $\overline{d_3^i} \neq\leftarrow$ by Inv2, thus $\overline{d_3^i} = \perp$. Consider the point when G_2 created $(3, \overline{x_3^i}, \overline{y_3^i})$. If G_2 was processing a shoot equivalent to $(3, \overline{y_3^i}, \{K^i, K'\})$, then we have $\overline{y_2^i} \in P_2^{-1}$ after this point, and the purported call to $\textsc{Adapt}(2, \overline{x_2^i}, \overline{y_2^i}, \ldots)$ would not have happened. Otherwise, after G_2 created $(3, \overline{x_3^i}, \overline{y_3^i})$, G_2 would detect a shoot formed by $(3, \overline{x_3^i}, \overline{y_3^i})$ and $(3, x_3^*, y_3^*)$ in the subsequent $\textsc{CollectTP}$-call and would have aborted;

(iii) If $\overline{n_3^i} \geq \mathit{cycleStartNum} > n_3^*$, then since it *cannot be* $\overline{y_3^i} \in EB(z')$, by Lemma 4 (i) and (ii) we have $\overline{d_3^i} =\leftarrow$. However, this along with $x_3^* \oplus \overline{x_3^i} = k^* \oplus k^i$ contradicts Inv3;

(iv) If $n_3^* \geq \mathit{cycleStartNum} > \overline{n_3^i}$, then: if $d_3^* =\leftarrow$ then it contradicts Inv3; otherwise, it contradicts Proposition 16 – here we note that the purported call to $\textsc{Adapt}(2, x_2^*, \overline{y_2^i}, \mathit{edir}^*, \mathit{enum}^*)$ may happen during G_2 processing a 13- or 31-TP, or during the *Fill-in-Rung-Phase* of an earlier $\textsc{ProcessShoot}$-call; however, in each case, the involved 3-query is "anchored" at the involved old E-chain, so that the argument carries.

By the above, the purported call to $\textsc{Adapt}(2, x_2^*, \overline{y_2^i}, \mathit{edir}^*, \mathit{enum}^*)$ would not have been possible. Thus $\overline{y_2^i} \notin P_2^{-1}$ is kept till the call to $\textsc{Adapt}(2, \overline{x_2^i}, \overline{y_2^i}, \ldots)$, and the latter would not cause abort.

Then consider the subsequent assertions. The first assertion causes abort if $\exists k' \neq k^i : \overline{x_2^i} \oplus k' \in P_1^{-1}$. This is not possible right after $\textsc{RandAssign}$ by (7), and would never hold till G_2 checking the assertion since no new 1-query is created during this (short) period. The second assertion states that if there exists some $k' \neq k^i$ and $(3, \overline{x_3^i}', \overline{y_3^i}', \overline{d_3^i}', \overline{n_3^i}')$ such that $\overline{x_3^i}' = \overline{y_2^i} \oplus k'$, then $\overline{y_3^i}' \notin \mathit{Border}$ and the 33-shoot $(3, \overline{y_3^i}', \{K^i, K'\})$ has been in $\mathit{ShootQueue}$. Note that this implies $\overline{x_3^i}' \oplus \overline{x_3^i} = k^i \oplus k'$, thus $\overline{n_3^i}' \geq \mathit{cycleStartNum}$, as otherwise it along with $\overline{n_3^i} < \mathit{cycleStartNum}$ implies $(3, K^i, \overline{x_3^i}) \in \mathit{Completed}$ before the cycle by Inv8 and Inv7, a contradiction. Moreover, $\overline{d_3^i}' \neq\leftarrow$, otherwise contradicts Inv3. As no 3-query with $\overline{d_3^i}' \in \{\rightarrow, \perp\}$ has been created in the current $\textsc{Process11Shoot}$-call, $(3, \overline{x_3^i}', \overline{y_3^i}')$ was created in an earlier chain-reaction call; and after creating $(3, \overline{x_3^i}', \overline{y_3^i}')$, G_2 would have detected $(3, \overline{y_3^i}', \{K^i, K'\})$ and pushed it into $\mathit{ShootQueue}$. Furthermore, by Proposition 8 we have $\textsc{DAwareness}(\overline{y_3^i}', Y3) = 1$, thus $\overline{y_3^i}' \notin \mathit{Border}$ by Lemma 11. Therefore, the second assertion never causes abort either.[21]

G_2 then iterates for all nodes $y'_{3,i}$. The non-abortion argument is similar to the above by symmetry.

G_2 Never Aborts During the Fill-in-Rung-Phase. In this phase, G_2 first iterates from $(x'_{3,t}, y'_{3,t})$ to $(x'_{3,1}, y'_{3,1})$. Consider an arbitrary pair $(x'_{3,i}, y'_{3,i})$ encountered in this iteration, and assume the query is $(3, x'_{3,i}, y'_{3,i}, d'_{3,i}, n'_{3,i})$. When G_2 finds a 2-query $(2, x_{2,2i}, y_{2,2i}, d_2, n_2)$ with $y_{2,2i} = x'_{3,i} \oplus k_1$, it would check an assertion, which fails if $(3, K_1, x'_{3,i}) \notin \mathit{Completed}$. We proceed to argue that this assertion never fails. For this, we argue that $n'_{3,i}, n_2 < \mathit{cycleStartNum}$, so that the claim holds by Inv6.

[21] In fact, we feel that such 3-queries $(3, \overline{x_3^i}', \overline{y_3^i}')$ cannot exist. But we cannot find a proof, so we take the current argument and implementation.

– Towards a contradiction, we first assume $n_2 > cycleStartNum$. Then there should have been a call to $\textsc{Adapt}(2, x_2^*, y_{2,2i}, edir^*, enum^*)$, as all the 2-queries newly created in this cycle are adapted ones. Assume that this call corresponds to the path

$$(K^*, k^*), (K^*, x_1^*, y_3^*), (1, x_1^*, y_1^*), (3, x_3^*, y_3^*, d_3^*, n_3^*),\ (k^* = y_1^* \oplus x_2^* = x_3^* \oplus y_{2,2i}),$$

then it holds $x_3^* \oplus x'_{3,i} = k^* \oplus k_1$.
We now show $n'_{3,i}, n_3^* < cycleStartNum$, thus $y_3^* \notin Border$ by Proposition 15,[22] and further $y_{2,2i} \in P_2^{-1}$ before the cycle by Inv8 and Inv7, and the purported $\textsc{Adapt}$-calls should not have happened. Wlog assume $n_3^* \geq cycleStartNum$ and $n_3^* > n'_{3,i}$. Then: if $d_3^* = \leftarrow$, then $x_3^* \oplus x'_{3,i} = k^* \oplus k_1$ is not possible by Inv3; otherwise, it contradicts Proposition 16 – similarly to the above argument for the *Shoot-Growing-Phase*, the involved 3-query $(3, x_3^*, y_3^*)$ is necessarily "anchored" at the involved old E-chain, so that the argument carries. Thus $n_2 < cycleStartNum$.
– We then assume $n'_{3,i} \geq cycleStartNum$. Then as the previous discussion establishes $n_2 < cycleStartNum$, we got $n'_{3,i} \geq cycleStartNum > n_2$, thus $d'_{3,i} \neq \leftarrow$ by Inv2. This however contradicts Proposition 16. Thus $n'_{3,i} < cycleStartNum$.

By the above, the first assertion never causes abort.

On the other hand, when G_2 finds $x'_{3,i} \oplus k_1 \notin P_2^{-1}$, it would evaluate in the (incomplete) chain corresponding to $(3, K_1, x'_{3,i})$, make a call to $\textsc{Adapt}(2, x_{2,2i}, y_{2,2i}, edir, enum)$, call $\textsc{UpdateCompleted}$, and finally check another assertion. We proceed to argue none of these three actions causes abortion. First, as argued, if $(3, K_1, x'_{3,i}) \notin Completed$, then $y_{2,2i} \notin P_2^{-1}$ necessarily holds right before this call. A similar argument could show that $x_{2,2i} \notin P_2$ also necessarily holds, thus this $\textsc{Adapt}$-call does not abort. As a consequence, the values in the chain would be consistent, and $\textsc{UpdateCompleted}$ does not abort either.

Second, the assertion fails if $\exists k \neq k_1, k_2 : x_{2,2i} \oplus k \in P_1^{-1}$ or $y_{2,2i} \oplus k \in P_3$. Assume otherwise, e.g. there exists a 3-query $(3, x_3^\circ, y_3^\circ, d_3^\circ, n_3^\circ)$ and $k^\circ \in \mathcal{Z} \setminus \{k_1, k_2\}$ such that $x_3^\circ \oplus k^\circ = y_{2,2i}$. Then it holds $x_3^\circ \oplus k^\circ = x'_{3,i} \oplus k_1$. But this is not possible, as we would exclude each possibility (similar to what we did for the *Shoot-Growing-Phase*):

(i) If $n_3^\circ, n'_{3,i} < cycleStartNum$, then $y'_{3,i} \notin Border$ by Proposition 15, and thus $y_{2,2i} \in P_2^{-1}$ before the cycle by Inv8 and Inv7, and the purported $\textsc{Adapt}$-calls should not have happened;
(ii) If $n_3^\circ, n'_{3,i} \geq cycleStartNum$, wlog assume $n_3^\circ > n'_{3,i}$, then $n_3^\circ > cycleStartNum$, thus $d_3^\circ = \perp$ by Inv2, and either the purported $\textsc{Adapt}$-call should not have happened, or G_2 would have aborted in an earlier $\textsc{CollectTP}$-call.
(iii) The case of $n'_{3,i} \geq cycleStartNum > n_3^\circ$ is excluded by an argument similar to the previous discussion on the query $(3, x_3^*, y_3^*)$;
(iv) If $n_3^\circ \geq cycleStartNum > n'_{3,i}$, then $d_3^\circ \neq \leftarrow$ by Inv2, thus $d_3^\circ \in \{\rightarrow, \perp\}$. From the code we know that right after $(2, x_{2,2i}, y_{2,2i}, \perp)$ is created, it holds $y_{2,2i} \oplus k_2 (= x'_{3,i} \oplus k_1 \oplus k_2) \in B_2(z)$. On the other hand, $x_3^\circ = x'_{3,i} \oplus k_1 \oplus k^\circ$; thus $(y_{2,2i} \oplus k_2) \oplus x_3^\circ = k_2 \oplus k^\circ \in 2\mathcal{Z}$. From Lemma 4 (ii) we know $x_3^\circ \in B_2(z)$ (note that when $n_3^\circ = cycleStartNum$ it holds $x_3^\circ = z$); then, by an argument similar to Proposition 17, we could reach a "pseudo-cycle" in B_2 and show that $(y_{2,2i} \oplus k_2) \oplus x_3^\circ = k_2 \oplus k^\circ$ is not possible.

G_2 then checks for $x'_{3,i} \oplus k_2$, and the involved argument is similar to the above. Furthermore, the argument for the second iteration also follows the same line. Thus the *Fill-in-Rung-Phase* would not cause abort.

G_2 Never Aborts During the Shoot-Completing-Phase. In this phase, G_2 first iterates from $(3, x'_{3,t}, y'_{3,t})$ and $(1, x'_{1,t}, y'_{1,t})$ to $(3, x'_{3,1}, y'_{3,1})$ and $(1, x'_{1,1}, y'_{1,1})$, and calls $\textsc{Adapt}(3, x_{3,i}, y_{3,i}, \perp, \perp)$ and $\textsc{Adapt}(1, x_{1,i}, y_{1,i}, \perp, \perp)$ for each of them.

We consider the involved "outer" values $y_{3,i}$ and $x_{1,i}$ first. As the $\textsc{Process11Shoot}$-call is safe (Proposition 21), when this call is made, it holds $xebval_l(K_2, K_1, x_1^\circ) = \perp$ for any $l \geq 2$. Thus the $2t - 1$ values $x_{1,t}, y_{3,t-1}, x_{1,t-1}, \ldots, y_{3,1}, x_{1,1}$ are all newly-obtained during the *Make-E-Chain-Phase*, and by Inv5, these values are not in P_1 and P_3^{-1} respectively.

We then consider the involved "inner" values $x_{3,i}$ and $y_{1,i}$. For each i, the value $y_{1,i}$ is computed from the corresponding 1-query $(1, x'_{1,i}, y'_{1,i}, d'_{1,i}, n'_{1,i})$ with $x'_{1,i}$ being a node of the involved old E-chain. We distinguish two cases:

[22] By Proposition 15, if $y_3^* \in Border$, then $(3, x_3^*, y_3^*, d_3^*, n_3^*)$ must be newly created in this $\textsc{Process11Shoot}$-call, and thus $n_3^* > cycleStartNum$.

(i) When $n'_{1,i} \geq cycleStartNum$, then $d'_{1,i} = \rightarrow$ by Proposition 16. Thus $y_{1,i} = y'_{1,i} \oplus k_1 \oplus k_2 \notin P_1^{-1}$ right after $(1, x'_{1,i}, y'_{1,i})$ is created. Note that according to Propositions 6, 18 and 19, if $y_{1,i}$ is pebbled at some later point (but earlier than PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_1, K_2)$ is called), then it's necessarily due to some earlier-processed shoots;

(ii) When $n'_{1,i} < cycleStartNum$, then $y'_{1,i} \oplus k_1 \oplus k_2 \notin P_1^{-1}$ holds before this simulator cycle, as otherwise $y_1^\circ \in P_1^{-1}$ would be inferred by Inv8 and Inv7, so that this call to PROCESS11SHOOT would not have happened. Similarly, according to Propositions 6, 18 and 19, if $y_{1,i} \in P_1^{-1}$ holds at some later point (but earlier than PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_1, K_2)$ is called), then it's necessarily due to some earlier-processed shoots.

Similar argument carries for each $x_{3,i}$.

We note that if G_2 has not aborted till the *Shoot-Completing-Phase*, then the $2t$ values $x_{3,i}$ and $y_{1,i}$ correspond to $yb2val_l(k_2, k_1, y_1^\circ)$ for $l = 1, \ldots, 2t$. More concretely, $x_{3,t} = yb2val_1(k_2, k_1, y_1^\circ)$, $y_{1,t} = yb2val_2(k_2, k_1, y_1^\circ)$, $x_{3,t-1} = yb2val_3(k_2, k_1, y_1^\circ)$, $y_{1,t-1} = yb2val_4(k_2, k_1, y_1^\circ)$, ...

Now we distinguish two possibilities, to figure out the PROCESSSHOOT-calls that happened earlier than PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_1, K_2)$ and may affect the $2t$ "inner" values:

(i) When $x_1^\circ = z'$ (note that the current simulator cycle is due to D querying $\mathrm{P}i^\delta(z) \rightarrow z'$), these earlier-processed shoots are of the form $(1, x_1^\circ, \{K_1, K_4\})$ and $(1, x_1^\circ, \{K_2, K_3\})$ for $K_3, K_4 \notin \{K_1, K_2\}$;

(ii) When $x_1^\circ \neq z'$, by Lemma 4 (i), the path between x_1° and z' is directed from z' to x_1°. This implies the existence of an E-query $(K^*, x_1^\circ, y_3^*, \leftarrow)$ for some y_3^*. Following the same line as the argument for Proposition 21, it can be seen the "interesting" earlier-processed shoots are of the form $(3, y_3^*, \{K^*, K_1\})$, $(3, y_3^*, \{K^*, K_2\})$, $(1, x_1^\circ, \{K_1, K_4\})$, or $(1, x_1^\circ, \{K_2, K_3\})$ with $K_3, K_4 \notin \{K_1, K_2\}$.

By an inspection of these cases and Proposition 19, it can be seen none of them can pebble $yb2val_l(k_2, k_1, y_1^\circ)$ for $l \geq 2$. By all the above, the ADAPT-calls for the $2t - 1$ pairs $(x_{1,t}, y_{1,t}), (x_{3,t-1}, y_{3,t-1}), (x_{1,t-1}, y_{1,t-1}), \ldots$ would not cause abort. More clearly:

(i) For $1 \leq i \leq t - 1$, G_2 necessarily finds $x_{3,i} \notin P_3$ and $y_{3,i} \notin P_3^{-1}$, thus calling ADAPT$(3, x_{3,i}, y_{3,i}, \perp, \perp)$, which would not cause abort;

(ii) For $1 \leq i \leq t$, G_2 necessarily finds $x_{1,i} \notin P_1$ and $y_{1,i} \notin P_1^{-1}$ and calls ADAPT$(1, x_{1,i}, y_{1,i}, \perp, \perp)$ which does not cause abort.

The pair $(x_{3,t}, y_{3,t})$ is not covered by the above analysis. For this pair, note that $y_{3,t} = xebval_1(K_2, K_1, x_1^\circ)$ and $x_{3,t} = yb2val_1(K_2, K_1, y_1^\circ)$. It can be seen that the earlier-processed shoots of the form $(1, x_1^\circ, \{K_2, K_3\})$ and $(3, y_3^*, \{K^*, K_2\})$ (y_3^* is the parent of x_1° in $EB(z')$) may affect the status of $y_{3,t}$ and $x_{3,t}$. We distinguish three cases:

(i) The earliest "interesting" shoot processed before PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_1, K_2)$ is $(1, x_1^\circ, \{K_2, K_3\})$. This implies the existence of an additional 1-query $(1, x_1^{\circ\circ\circ}, y_1^{\circ\circ\circ}, d_1^{\circ\circ\circ}, n_1^{\circ\circ\circ})$ (with $y_1^\circ \oplus y_1^{\circ\circ\circ} = k_2 \oplus k_3$) besides $(1, x_1^{\circ\circ}, y_1^{\circ\circ}, d_1^{\circ\circ}, n_1^{\circ\circ})$ with $y_1^{\circ\circ} \oplus y_1^\circ = k_1 \oplus k_2$, the one that helps form the 11-shoot $(1, x_1^\circ, \{K_1, K_2\})$ (so that G_2 could detect $(1, x_1^\circ, \{K_2, K_3\})$ after creating $(1, x_1^\circ, y_1^\circ)$). These imply $y_1^{\circ\circ\circ} \oplus y_1^{\circ\circ} = k_1 \oplus k_3$. Furthermore, both $(1, x_1^{\circ\circ}, y_1^{\circ\circ})$ and $(1, x_1^{\circ\circ\circ}, y_1^{\circ\circ\circ})$ already exist before this cycle, as otherwise G_2 would have aborted in COLLECTTP$(1, x_1^\circ, y_1^\circ)$. Also, they cannot be in *Border*, otherwise contradicting Lemma 11 and Proposition 8. Thus by Inv8 and Inv7, two paths

$$(K_1, k_1), (K_1, x_1^{\circ\circ}, y'_{3,t}), (1, x_1^{\circ\circ}, y_1^{\circ\circ}), (2, x_2^\circ, y_2^\circ), (3, x'_{3,t}, y'_{3,t})\ (y_1^{\circ\circ} \oplus x_2^\circ = y_2^\circ \oplus x'_{3,t} = k_1)$$

and

$$(K_3, k_3), (K_3, x_1^{\circ\circ\circ}, y_3^{\circ\circ\circ}), (1, x_1^{\circ\circ\circ}, y_1^{\circ\circ\circ}), (2, x_2^\circ, y_2^\circ), (3, x_3^{\circ\circ\circ}, y_3^{\circ\circ\circ})\ (y_1^{\circ\circ\circ} \oplus x_2^\circ = y_2^\circ \oplus y_3^{\circ\circ\circ} = k_3)$$

exist in the history, cf. Fig. 10 (left).[23] We note it holds $x_{3,t} = x'_{3,t} \oplus k_1 \oplus k_2 = x_3^{\circ\circ\circ} \oplus k_2 \oplus k_3$.
Then, G_2 popping $(1, x_1^\circ, \{K_2, K_3\})$ leads to a call to PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_2, K_3)$. Inside this call, G_2 would find $x_{3,t} \notin P_3$ and $y_{3,t} \notin P_3^{-1}$ – as we argued for $(x_{3,i}, y_{3,i})$, and since we assumed $(1, x_1^\circ, \{K_2, K_3\})$

[23] Note that the query $(3, x'_{3,t}, y'_{3,t})$ is consistent with the notations for PROCESS11SHOOT$(x_1^\circ, y_1^\circ, K_1, K_2)$.

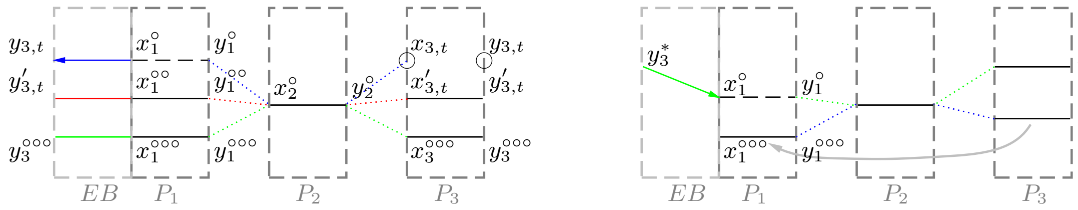


Fig. 10. (Left) For case (i): the two involved completed paths and the two 11-shoots. The lines in red, blue, and green indicate E-queries labeled K_1, K_2, and K_3 respectively, while the colored dotted lines indicate the connection under the corresponding round-keys. (Right) For case (ii): implying the existence of $(1, x_1^{\circ\circ\circ}, y_1^{\circ\circ\circ})$. The lines in blue and green indicate connections under (K_2, k_2) and (K^*, k^*) respectively.

is the "earliest interesting" shoot. Thus G_2 would make a non-aborting call to $\text{ADAPT}(3, x_{3,t}, y_{3,t}, \perp, \perp)$. After this adaptation, the following path exists in the history:

$$(K_2, k_2), (K_2, x_1^\circ, y_{3,t}), (1, x_1^\circ, y_1^\circ), (2, x_2^\circ, y_2^\circ), (3, x_{3,t}, y_{3,t}), \text{ with } y_1^\circ \oplus x_2^\circ = y_2^\circ \oplus x_{3,t} = k_2.$$

Moreover, G_2 would add the tuple $(3, y_{3,t}, \{K_2, K_3\})$ to *ProcessedShoots*. Therefore, later inside the call to $\text{PROCESS11SHOOT}(x_1^\circ, y_1^\circ, K_1, K_2)$, when G_2 iterates for $(x_{3,t}, y_{3,t})$, it indeed finds both $P_3(x'_{3,t} \oplus k_1 \oplus k_2) = y_{3,t}$ and $\exists (3, y_{3,t}, \{K_2, K_3\}) \in \textit{ProcessedShoots}$ hold ($(3, y_{3,t}, \{K_2, K_3\})$ is equivalent to the processed $(1, x_1^\circ, \{K_2, K_3\})$). Thus in this case, when $\text{ADAPT}(3, x_{3,t}, x_{3,t}, \perp, \perp)$ is not called in $\text{PROCESS11SHOOT}(x_1^\circ, y_1^\circ, K_1, K_2)$, the involved assertions would not cause abort;

(ii) The earliest "interesting" shoot processed before $\text{PROCESS11SHOOT}(x_1^\circ, y_1^\circ, K_1, K_2)$ is $(i, z^\circ, \{K^*, K_2\})$ which is equivalent to $(3, y_3^*, \{K^*, K_2\})$. This implies the existence of an additional 1-query $(1, x_1^{\circ\circ\circ}, y_1^{\circ\circ\circ})$ with $y_1^\circ \oplus y_1^{\circ\circ\circ} = k_2 \oplus k^*$ when G_2 is to create $(1, x_1^\circ, y_1^\circ)$, cf. Fig. 10 (right). Then one can see that the analysis for this case has no essential difference with the analysis for case (i), leading to the same conclusion: in this case, when $\text{ADAPT}(3, x_{3,t}, x_{3,t}, \perp, \perp)$ is not called in $\text{PROCESS11SHOOT}(x_1^\circ, y_1^\circ, K_1, K_2)$, the involved assertions would not cause abort;

(iii) Otherwise, $x_{3,t} \notin P_3$ and $y_{3,t} \notin P_3^{-1}$ would be kept till $\text{PROCESS11SHOOT}(x_1^\circ, y_1^\circ, K_1, K_2)$, and G_2 would make a non-aborting call to $\text{ADAPT}(3, x_{3,t}, y_{3,t}, \perp, \perp)$ similarly as described.

Finally, for each $(x_{1,i}, y_{1,i})$, if $\text{ADAPT}(1, x_{1,i}, y_{1,i}, \perp, \perp)$ is called, then G_2 would check two additional groups of assertions. The first assertion states that this newly created AD-1-query would not form any 31-TPs that makes sense (i.e. not in a completed path): it fails, if there exists $K \notin \{K_1, K_2\}$ such that $\textit{ETable}[K](x_{1,i}) \in P_3^{-1}$ but $(1, K, x_{1,i}) \notin \textit{Completed}$. This assertion clearly never fails, because if $x_{1,i} \in \textit{ETable}[K]$ then the corresponding E-query was necessarily created in an earlier chain-reaction call in this cycle, after which the E-query is indeed in a completed path by Lemma 3. The second group of assertions are in the subsequent call to UPDATECOMPLETED, which never fail because the previous call to $\text{ADAPT}(1, x_{1,i}, y_{1,i}, \perp, \perp)$ succeeds. Similar assertions exist for each $(x_{3,i}, y_{3,i})$, and the non-abortion argument is similar by symmetry.

By all the above, the assertions and adaptations in the iteration from $(3, x'_{3,t}, y'_{3,t})$ and $(1, x'_{1,t}, y'_{1,t})$ to $(3, x'_{3,1}, y'_{3,1})$ and $(1, x'_{1,1}, y'_{1,1})$ would not cause abort. G_2 then iterates from $(3, x'_{3,t+1}, y'_{3,t+1})$ and $(1, x'_{1,t+2}, y'_{1,t+2})$ to $(3, x'_{3,2t}, y'_{3,2t})$ and $(1, x'_{1,2t+1}, y'_{1,2t+1})$ and calls $\text{ADAPT}(3, x_{3,i}, y_{3,i}, \perp, \perp)$ and $\text{ADAPT}(1, x_{1,i}, y_{1,i}, \perp, \perp)$ for each of them. The argument for this iteration is similar to the previous one by symmetry. These complete the proof. □

Concluding. We conclude with the following lemma.

Lemma 15. *The adaptations and assertions in a simulator cycle induced by D making* $\text{P1}^{-1}(y_1)$ *or* $\text{P3}(x_3)$ *never cause abort.*

Proof. By the pseudocode, adaptations and assertions only occur in the subsequent calls to COLLECTTP, PROCESSSHOOT, PROCESS21TP, and PROCESS23TP, the non-abortions of which have been established by Propositions 17, 23, and 22 respectively. □

9 Termination

Recall from subsection 6.1 that D makes q_e, q_h, and q_p queries to $\mathrm{E}/\mathrm{E}^{-1}$, H, and $\mathrm{P}i/\mathrm{P}i^{-1}$ respectively. We further assume that D makes q_{p_1}, q_{p_2}, and q_{p_3} queries to $\mathrm{P1}/\mathrm{P1}^{-1}$, $\mathrm{P2}/\mathrm{P2}^{-1}$, and $\mathrm{P3}/\mathrm{P3}^{-1}$ respectively $(q_{p_1}+q_{p_2}+q_{p_3}=q_p)$. Then we have:

(i) $|HQueries| = |\mathcal{Z}| \le q_h$, as $|HQueries|$ only increasing by at most 1 per D's query to H;

(ii) The number of detected 13-, 31-, and H-TPs is at most q_e in total;
  - Following the idea of Coron et al. [CPS08,HKT11]: by Proposition 3, the number of such TPs does not exceed the number of E-queries made by D.

(iii) The number of detected 12-, 32-, and MidTPs is at most q_{p_2} in total.
  - Consider D querying $\mathrm{P2}(x_2)$; for $\mathrm{P2}^{-1}(y_2)$ the discussion is similar by symmetry. If there exists a 1-query $(1,x_1,y_1)$ meets $y_1 = x_2 \oplus k$ and $x_1 \notin ETable[K]$, then only one 12-TP would be processed, and G_2 would finally create an AD-2-query $(2,x_2,y_2,\perp)$, which is unable to help form MidTPs by Proposition 4. Otherwise, G_2 would create a 2-query $(2,x_2,y_2,\rightarrow)$, which can be involved in a MidTP. However, by Proposition 5 we know the 2-query created in this case would help form at most one MidTP. The two cases are mutual exclusive, thus the total number never exceeds q_{p_2}.

ProcessShoot-calls clearly contribute to $|Queries|$ and $|EQueries|$ a lot, and we should bound the number of such calls. This task is a bit harder. It relies on two propositions.

Proposition 24. *During any long simulator cycle, for each newly created 1-query $(1,x_1,y_1,d)$ with $d=\leftarrow$ or $\perp$, the number of 11-shoots $(1,x_1,\{K,K'\}) \notin ProcessedShoots$ that can be formed by $(1,x_1,y_1,d)$ does not exceed twice the number of earlier P-cycles. Similar claim holds for each newly created 3-query $(3,x_3,y_3,d)$ with $d=\rightarrow$ or $\perp$.*

Proof. Wlog consider $(1,x_1,y_1,d)$. If the claimed bound does not hold, then there necessarily exists three 1-queries $(1,x'_1,y'_1,d')$, $(1,x''_1,y''_1,d'')$, and $(1,x'''_1,y'''_1,d''')$, such that for $k_1,k_2,k_3,k_4,k_5,k_6 \in \mathcal{Z}$ it holds:

- $y_1 = y'_1 \oplus k_1 \oplus k_2$, and $y_1 = y''_1 \oplus k_3 \oplus k_4$, and $y_1 = y'''_1 \oplus k_5 \oplus k_6$, and
- $(1,x'_1,y'_1,d')$, $(1,x''_1,y''_1,d'')$, and $(1,x'''_1,y'''_1,d''')$ are created in the same simulator cycle.

Note that the three queries were necessarily created in a long cycle. Assume that the cycle is induced by D querying $\mathrm{P}i^{\delta}(z) \rightarrow z'$ $((i,\delta) \in \{(1,-),(3,+)\})$. As argued, we have $d',d'',d''' = \rightarrow$ or $\perp$. However, if two of them equal $\rightarrow$, then it would contradict Inv3; thus at least two of them equal $\perp$. Wlog assume $d',d''=\perp$. Then by Lemma 4 (ii) we have $y'_1,y''_1 \in B_2(z)$. Thus we got a "pseudo-cycle" $z - \ldots - y'_1 \xrightarrow{\oplus k_1 \oplus k_2 \oplus k_3 \oplus k_4} y''_1 - \ldots - (z)$ in B_2, which would ultimately contradict Lemma 6, and thus G_2 would have aborted in this earlier cycle of $\mathrm{P}i^{\delta}(z) \rightarrow z'$. Therefore, it is not possible for a 1-query forming unprocessed shoots with more than two 1-queries created in the same cycle.

Then, as 1- and 3-queries cannot be created in E- nor H-cycles, we reach the claim. □

Proposition 25. *During any long cycle, two distinct newly created 1-queries $(1,x_1,y_1,d)$ and $(1,x'_1,y'_1,d')$ $(d,d' \in \{\leftarrow,\perp\})$ cannot both form unprocessed shoots (shoots not in ProcessedShoots) with the 1-queries created in the same earlier cycle.*

Proof. Consider 1-queries first. Assume that the long cycle creating $(1,x_1,y_1,d)$ and $(1,x'_1,y'_1,d')$ is induced by D querying $\mathrm{P}i^{\delta}(z) \rightarrow z'$ $((i,\delta) \in \{(1,-),(3,+)\})$. By Lemma 4 (ii) we have $y_1,y'_1 \in B_2(z)$.

Then, towards a contradiction, assume that there are two 1-queries $(1,x''_1,y''_1,d'')$ and $(1,x'''_1,y'''_1,d''')$ and $k_1,k_2,k_3,k_4 \in \mathcal{Z}$ such that:

- $y_1 = y''_1 \oplus k_1 \oplus k_2$, and $y'_1 = y'''_1 \oplus k_3 \oplus k_4$, and
- $(1,x''_1,y''_1,d'')$ and $(1,x'''_1,y'''_1,d''')$ are created in the same (necessarily long, as argued) simulator cycle.

Assume that the long cycle creating $(1,x''_1,y''_1,d'')$ and $(1,x'''_1,y'''_1,d''')$ is induced by D querying $\mathrm{P}i^{\delta}(z^{\circ}) \rightarrow z^{\circ\circ}$ $((i,\delta) \in \{(1,-),(3,+)\})$. As argued, we have $d'',d''' = \rightarrow$ or $\perp$. It falls into three cases:

Case 1: $(1,x''_1,y''_1,d'') = (1,x'''_1,y'''_1,d''')$. In this case, $y_1 = y''_1 \oplus k_1 \oplus k_2$ and $y'_1 = y''_1 \oplus k_3 \oplus k_4$ would imply $y_1 \oplus y'_1 = k_1 \oplus k_2 \oplus k_3 \oplus k_4 \in 4\mathcal{Z}$, and by $y_1,y'_1 \in EB(z)$ (Lemma 4 (ii)) we got a "pseudo-cycle" $z - \ldots - y_1 \xrightarrow{k_1 \oplus k_2 \oplus k_3 \oplus k_4} y'_1 - \ldots - (z)$ in B_2, so that G_2 necessarily aborts before it completes creating the two 1-queries $(1,x_1,y_1,d)$ and $(1,x'_1,y'_1,d')$.

Case 2: During the earlier long cycle, $(1, x_1'', y_1'', d'')$ and $(1, x_1''', y_1''', d''')$ are in the same shoot. Then there exists a ProcessShoot-call such that $(1, x_1'', y_1'', d'')$ is "anchored" at its old E-chain while $(1, x_1''', y_1''', \perp)$ at its new E-chain (or may be opposite; this does not matter), and $y_1'' \oplus y_1''' = k_5 \oplus k_6$ for $k_5, k_6 \in \mathcal{Z}$. By Proposition 16, it has to be $d'' = \rightarrow$, otherwise $(1, x_1'', y_1'')$ was not created in the same cycle as $(1, x_1''', y_1''')$.

Now, as $y_1, y_1' \in B_2(z)$, either the path between z and y_1 is directed from z to y_1, or the path between z and y_1' is from z to y_1' (it's not hard to see this holds even if z equals y_1 or y_1').

First, assume that the path $z \rightarrow \ldots \rightarrow y_1$ is to y_1. This implies the existence of a 2-edge $(y_1, x_3, k^*, \leftarrow)$ in B_2. This implies the existence of a 2-query $(2, x_2^*, y_2^*, d_2^*)$ with $x_2^* = y_1 \oplus k^*$, $y_2^* = x_3 \oplus k^*$, and $d_2^* \neq \rightarrow$. Depending on d_2^*, we distinguish two sub-cases:

Sub-case 2.1: $d_2^* = \leftarrow$, i.e. (y_1, x_3, k^*) is an <u>RA-2-edge</u>. As $y_1 = y_1'' \oplus k_1 \oplus k_2$, we got $x_2^* \oplus y_1'' = k^* \oplus k_1 \oplus k_2 \in 3\mathcal{Z}$. Thus $(1, x_1'', y_1'', \rightarrow)$ and $(2, x_2^*, y_2^*, \leftarrow)$ would contradict Inv2, cf. Fig. 11 (left).

Sub-case 2.2: $d_2^* = \perp$, i.e. (y_1, x_3, k^*) is an <u>AD-2-edge</u>. Assume that the mirror E-query of $(2, x_2^*, y_2^*, \perp)$ is $(K^{**}, x_1^{**}, y_3^{**}, \leftarrow)$. These imply the existence of another 1-query $(1, x_1^{**}, y_1^{**}, d_1^{**})$ with $y_1^{**} = x_2^* \oplus k^{**} = y_1 \oplus k^* \oplus k^{**}$ (thus $y_1^{**} \oplus y_1'' \in 4\mathcal{Z}$). As argued (cf. *Case 2* in the proof of Lemma 6), since $(1, x_1^{**}, y_1^{**})$ lies between the heads of $(K^{**}, x_1^{**}, y_3^{**})$ and (y_1, x_3, k^*), it must be $d_1^{**} = \rightarrow$; this along with $(1, x_1'', y_1'', \rightarrow)$ would contradict Inv3, cf. Fig. 11 (right).

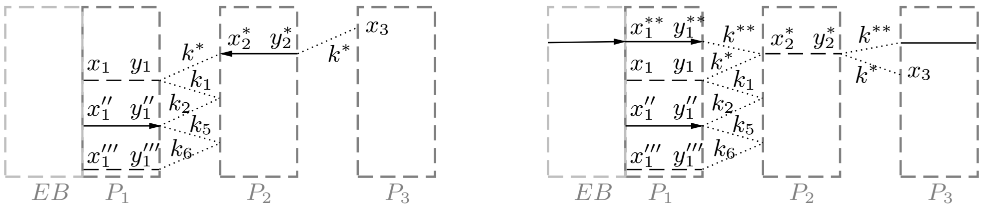


Fig. 11. For Proposition 25. Arrowed lines indicate directed queries, with arrows consistent with directions; dashed lines indicate AD-queries. (left) *sub-case 2.1*: (y_1, x_3, k^*) is an RA-2-edge; (right) *sub-case 2.2*: (y_1, x_3, k^*) is an AD-2-edge.

The above shows the contradiction based on the assumption that the path $z \rightarrow \ldots \rightarrow y_1$ is to y_1. For the other case, i.e . the path $z \rightarrow \ldots \rightarrow y_1'$ is from z to y_1', there is no essential difference (except for replacing $3\mathcal{Z}$ and $4\mathcal{Z}$ in *sub-case 2.1* and *2.2* by $5\mathcal{Z}$ and $6\mathcal{Z}$ respectively). In each subcase, G_2 cannot complete creating $(1, x_1, y_1, d)$ and $(1, x_1', y_1', d')$. These complete the analysis for *Case 2*.

Case 3: During the earlier long cycle, $(1, x_1'', y_1'', d'')$ and $(1, x_1''', y_1''', d''')$ are in two different shoots. We exclude three possibilities (which are a bit similar to *Sub-case 4.1.2* in the proof of Proposition 13) to show that G_2 cannot complete creating $(1, x_1, y_1, d)$ and $(1, x_1', y_1', d')$:

- If both d'' and d''' equal $\perp$, then by Lemma 4 (ii) we have $y_1'', y_1''' \in B_2(z^\circ)$. Thus we got a "pseudo-cycle" $z - \ldots - y_1 \overset{\oplus k_1 \oplus k_2}{\overline{\quad\quad}} y_1'' - \ldots - z^\circ - \ldots - y_1''' \overset{\oplus k_3 \oplus k_4}{\overline{\quad\quad}} y_1' - \ldots - (z)$ in B_2, which would finally contradict Lemma 6;
- If both d'' and d''' equal $\rightarrow$, then by construction, in this earlier-long cycle, G_2 necessarily created two AD-1-queries $(1, x_1^\circ, y_1^\circ, \perp)$ and $(1, x_1^{\circ\circ}, y_1^{\circ\circ}, \perp)$ with $y_1^\circ = y_1'' \oplus k_5 \oplus k_6$ and $y_1^{\circ\circ} = y_1''' \oplus k_7 \oplus k_8$ for $k_5, k_6, k_7, k_8 \in \mathcal{Z}$. By Lemma 4 (ii) we have $y_1^\circ, y_1^{\circ\circ} \in B_2(z^\circ)$, thus we got a "pseudo-cycle" $z - \ldots - y_1 \overset{\oplus k_1 \oplus k_2 \oplus k_5 \oplus k_6}{\overline{\quad\quad\quad\quad}} y_1^\circ - \ldots - z^\circ - \ldots - y_1''' \overset{\oplus k_3 \oplus k_4 \oplus k_7 \oplus k_8}{\overline{\quad\quad\quad\quad}} y_1' - \ldots - (z)$ in B_2, which would finally contradict Lemma 6;
- The "hybrid case": e.g. if $d'' = \rightarrow$ while $d''' = \perp$, then there exists $(1, x_1^\circ, y_1^\circ, \perp)$ such that $y_1^\circ = y_1'' \oplus k_5 \oplus k_6$ for $k_5, k_6 \in \mathcal{Z}$, and we got a "pseudo-cycle" $z - \ldots - y_1 \overset{\oplus k_1 \oplus k_2 \oplus k_5 \oplus k_6}{\overline{\quad\quad\quad\quad}} y_1^\circ - \ldots - z^\circ - \ldots - y_1''' \overset{\oplus k_3 \oplus k_4}{\overline{\quad\quad}} y_1' - \ldots - (z)$ in B_2.

These complete the analysis for 1-queries. For 3-queries the argument is similar by symmetry. □

Therefore, we reach the main claim:

Lemma 16. *The number of* Process11Shoot*-calls (*Process33Shoot*-calls, resp.) that appear in the l-th P-cycle is at most $2(l-1)$. As a consequence, In any G_2 execution, the number of* ProcessShoot*-calls is at most $2q_p^2$.*

Proof. It can be seen that if the l-th P-cycle is not a long one, then there will be no ProcessShoot-call. Otherwise, assume that $2(l-1)+1$ Process11Shoot-calls – in other words, $2(l-1)+1$ unprocessed 11-shoots – appear in a long cycle. By Inv1, distinct 11-shoots are necessarily formed by distinct pairs of 1-queries (cf. the analysis in subsection 4.5). By Proposition 24, these pairs are necessarily formed by $\lceil\frac{2(l-1)+1}{2}\rceil = l$ distinct 1-queries with $dir \in \{\leftarrow, \perp\}$ newly created in this cycle. However, by Proposition 25 we know any two distinct 1-queries among these l ones cannot form unprocessed shoots with the 1-queries created in the same earlier P-cycle. Thus there necessarily exist l earlier P-cycles, a contradiction. For Process33Shoot-calls the argument is similar by symmetry. The maximum number of long cycles is $q_{p_1} + q_{p_3} \le q_p$. Therefore, the total number of Process11Shoot-calls is at most $\sum_{l=1}^{q_p}(2l-2) \le q_p^2$; the same bound for Process33Shoot-calls, thus the claimed $2q_p^2$. ◻

The above culminate with the following bounds.

Lemma 17. *Let $\mu = (q_e + q_p)\cdot(q_p)^2$. Then in any execution D^{G_2}, it holds:*

(i) $|P_1|, |P_3| \le 13\mu$, $|P_2| \le 9\mu$, $|EQueries| \le q_e + q_p + 16\mu$, and the number of 11-shoots (33-shoots, resp.) in $DUShoots$ is at most 4μ;

(ii) the number of distinct calls to Check *is at most $169 q_h \mu^2$.*

Proof. Assuming $q_e + q_p \ge 4$, then $t \le \frac{q_e+q_p+4}{2} \le q_e + q_p$ (recalling from subsection 4.3 for the parameter t). For $q_e + q_p = 3$ it also holds $t = \frac{3+3}{2} \le q_e + q_p$.

Then we derive the bounds one-by-one. 2-queries can be created in three cases:

(i) D queries P2 or P2^{-1} – $\le q_{p_2} \le q_p$;
(ii) G_2 processing 13-, 31-, or H-TPs – $\le q_e$;
(iii) ProcessShoot-calls. Each such call creates at most $4t \le 4(q_e + q_p)$ 2-queries, while the number of such calls is at most $2q_p^2$ by Lemma 16.

In total, $|P_2| \le q_e + q_p + 8\mu \le 9\mu$ assuming $q_p \ge 1$. Clearly, it still holds when $q_p = 0$ (in this case $|P_2| = 0$).

We note that once $|P_1|$ or $|P_3| > 13\mu$, then G_2 would abort in AddQuery. However, our argument does not take advantage of this condition; thus it implies G_2 never aborts due to $|P_1|$ or $|P_3|$ exceeds. Our argument is as follows: 1-queries can be created in three cases

(i) D queries P1 or P1^{-1} – $\le q_{p_1}$;
(ii) G_2 processing 12-, 32-, or MidTPs – $\le q_{p_2}$;
(iii) ProcessShoot-calls. Each such call creates at most $4t+2$ 1-queries, thus in total it's $(4t+2)\cdot 2q_p^2 = (8t+4)q_p^2 \le 12\mu$.

They sum to $\le q_p + 12\mu \le 13\mu$ (similarly to $|P_2|$, regardless of $q_p = 0$ or not). The argument for $|P_3|$ is similar by symmetry.

E-queries can be created in three cases:

(i) D queries E or E^{-1} – $\le q_e$;
(ii) G_2 processing 12-, 32-, or MidTPs – $\le q_{p_2}$;
(iii) ProcessShoot-calls. Each such call creates at most $8t$ E-queries, thus in total it's 16μ.

In total $\le q_e + q_p + 16\mu$.

Each ProcessShoot-call adds at most $2t$ 11-shoots to $DUShoots$ – note that there are, indeed, $2t+1$ 11-shoots involved in each such call; however, by Proposition 8 and the pseudocode, at least one of them cannot be in $DUShoots$. Thus the number of 11-shoots in $DUShoots$ is at most 4μ. For 33-shoots in $DUShoots$ it's similar.

The above claims assume $q_e + q_p \ge 3$. One could check that when $q_e + q_p \le 2$, the bounds still hold: when $q_e + q_p = 1$, then only one set among $EQueries, P_1, P_2, P_3$ gets an element; when $q_e + q_p = 2$, it's not hard to see one of the best choices is to make 2 queries to P1^{-1} to induce 2 calls to Process11Shoot, and the resulted "real sizes" do not exceed our "claimed bounds", cf. Table 2.

Finally, the number of distinct Check-calls is at most $q_h \cdot |P_1| \cdot |P_3| \le q_h \cdot (13\mu)^2$, unless either $|P_1|$ or $|P_3|$ goes beyond 13μ. However, in that case, G_2 immediately aborts, and no more Check-calls can occur. ◻

The bound $O(q_p^2)$ given by Lemma 16 is clearly tight, as it can be matched by a very simple attack. Consequently, $|EQueries| = O(q^3)$ also seems tight. However, the tightness of all the other bounds remain unclear. Since the current simulator design has been extremely complicated, we defer the seeking for tight bounds for future.

Based on the above bounds, in the next section we bound the abort probability of G_2.

Table 2. For Lemma 17: cases of $q_e + q_p = 2$.

Case	t	$\lvert EQueries\rvert$	$\lvert P_1\rvert$	$\lvert P_2\rvert$	$\lvert P_3\rvert$	11-shoots in $DUShoots$	33-shoots in $DUShoots$
"real"	3	24	14	12	12	6	6
"claimed"	3	130	100	68	100	32	32

10 Abort-Probability of G_2

We first consider early-abortions, and then the CheckDUnaware-calls. As proved in Lemmata 12-15, these constitute all the abortions in G_2 executions.

Lemma 18. *In D^{G_2}, the probability of early-abortion is at most* $\frac{(1462+2144q_h^6)\cdot(q_e+q_p)^2\cdot q_p^4+2q_e^2+q_h^2+q_h^4}{N}$.

Proof. Consider a pair of queries $((K,x_1,y_3),(1,x_1,y_1))$. If the last call before this pair (logically) exist is $\text{EIn}^{-1}(K,y_3)$ or $\text{P1}^{-1}(y_1)$, then G_2 would abort. The number of such pairs is at most $|EQueries|\cdot|P_3|$, while the probability for G_2 to abort on a single pair is at most $\frac{1}{N-\text{Max}\{|EQueries|,|P_1|\}} \le \frac{1}{N-|EQueries|}$, thus the bound in total is $\frac{|EQueries|\cdot|P_3|}{N-|EQueries|} \le \frac{13\mu\cdot(q_e+q_p+16\mu)}{N-|EQueries|} \le \frac{221\mu^2}{N-|EQueries|}$ (holds even if $q_p = 0$). Similarly, the probability of abortion due to pairs of the form $((K,x_1,y_3),(3,x_3,y_3))$ is at most $\frac{|EQueries|\cdot|P_1|}{N-|EQueries|} \le \frac{221\mu^2}{N-|EQueries|}$.

Consider a pair of queries $((K,x_1,y_3),(K',x_1,y_3'))$. If the last call before this pair (logically) exist is $\text{EIn}^{-1}(K,y_3)$ or $\text{EIn}^{-1}(K',y_3')$, then G_2 would abort. The probability for a single pair is at most $\frac{1}{N-|EQueries|}$. For a pair of queries $((K,x_1,y_3),(K',x_1',y_3))$, if the last call before this pair (logically) exist is $\text{EIn}(K,x_1)$ or $\text{EIn}(K',x_1')$, then G_2 would abort. However, the two types of bad cases are mutual exclusive, thus the bound in total is $\frac{|EQueries|^2}{N-|EQueries|} \le \frac{(q_e+q_p+16\mu)^2}{N-|EQueries|}$. When $q_p \ge 1$, $q_e+q_p \le \mu$, and we got $\frac{289\mu^2}{N-|EQueries|}$; when $q_p = 0$, it's clearly $\frac{q_e^2}{N-|EQueries|}$. Thus the bound in total is $\frac{q_e^2+289\mu^2}{N-|EQueries|}$.

Consider a triple $(z,(i,x_i,y_i),(i,x_i',y_i'))$ with $z \in 6\mathcal{Z}$ and $y_i \oplus y_i' = z$. If the last call before this triple (logically) exist is $\text{RandAssign}(i,x_i,+)$, or $\text{RandAssign}(i,x_i',+)$, or H, then G_2 would abort. The probability is at most $\frac{q_h^6\cdot|P_i|^2}{N-|P_i|}$ in total. Similarly for triples $(z,(i,x_i,y_i),(i,x_i',y_i'))$ with $z \in 6\mathcal{Z}$ and $x_i \oplus x_i' = z$, thus the bound in total is $\sum_{i=1,2,3}\frac{2\cdot q_h^6\cdot|P_i|^2}{N-|P_i|} \le \frac{838q_h^6\mu^2}{N-|P_1|}$. As $0 \in 6\mathcal{Z}$, these already include the events $z' \in P_i^{-1}$ in $\text{RandAssign}(i,z,+)$ and $z' \in P_i$ in $\text{RandAssign}(i,z,-)$.

Consider a triple $(z,(1,x_1,y_1),(2,x_2,y_2))$ with $z \in 5\mathcal{Z}$ and $y_1 \oplus x_2 = z$. If the last call before this triple (logically) exist is $\text{RandAssign}(1,x_1,+)$, or $\text{RandAssign}(2,y_2,-)$, or H, then G_2 would abort. The probability is at most $\frac{q_h^5\cdot|P_1|\cdot|P_2|}{N-|P_1|}$ in total (as $|P_1| \ge |P_2|$). Similarly for triples $(z,(2,x_2,y_2),(3,x_3,y_3))$ with $z \in 5\mathcal{Z}$ and $y_2 \oplus x_3 = z$, thus in total $\frac{2\cdot q_h^5\cdot|P_1|\cdot|P_2|}{N-|P_1|} \le \frac{234q_h^5\mu^2}{N-|P_1|}$ (the upper bound on $|P_1|$ equals that on $|P_3|$).

Finally, in H, there are two other types of abortion, i.e. $Pr[\exists K_1 \ne K_2 : \mathbf{R}.\text{H}(K_1) = \mathbf{R}.\text{H}(K_2)] \le \frac{q_h^2}{N}$ and $Pr[\exists K_1 \ne K_2 \ne K_3 \ne K_4 : \bigoplus_{i=1,2,3,4}\mathbf{R}.\text{H}(K_i) = 0] \le \frac{q_h^4}{N}$. Thus assuming $|P_1| \le 13(q_e+q_p)\cdot q_p^2 \ll \frac{N}{2}$, $|EQueries| \le q_e+q_p+16(q_e+q_p)\cdot q_p^2 \ll \frac{N}{2}$, and substituting μ by $(q_e+q_p)\cdot q_p^2$, we reach the bound

$$\frac{(1462+2144q_h^6)\cdot(q_e+q_p)^2\cdot q_p^4+2q_e^2+q_h^2+q_h^4}{N}.$$

□

For clearness, we use a sub-claim for CheckDUnaware-calls.

Proposition 26. *A call to* CheckDUnaware *aborts with probability at most* $\frac{8q_h^2\mu}{N-q_e-q_p-16\mu}$.

Proof. Consider a call to $\text{CheckDUnaware}(x_1^\circ, X1)$ first. It is necessarily made due to D querying $\text{E}(K^\circ,x_1^\circ)$ or $\text{P1}(x_1^\circ)$. Consider an arbitrary tuple $(1,\{(x_1,y_1),(x_1',y_1')\}) \in DUShoots$. By Proposition 7, (a) there exists two E-queries $(K,x_1,y_3,\leftarrow)$ and $(K',x_1',y_3',\leftarrow)$ in $EQueries$ for some K,K',y_3,y_3'; (b) wlog we could assume that two 1-queries $(1,x_1,y_1,\rightarrow)$ and $(1,x_1',y_1',\perp)$ are in $Queries$, with $y_1 \oplus y_1' = k \oplus k'$. By these, before the call to $\text{CheckDUnaware}(x_1^\circ, X1)$ is made, G_2 has queried $\mathbf{E}.\text{E}^{-1}(K,y_3) \to x_1$, $\mathbf{E}.\text{E}^{-1}(K',y_3') \to x_1'$, and $\mathbf{R}.\text{P1}(x_1) \to y_1$. Since these three values are all in $DUShoots$, based on the queries not in $DUShoots$ (note that by design, this already includes all the earlier query-answer pairs obtained by D) and the new query from D, the three values x_1', x_1, and y_1 cannot be determined; thus they remain fresh when $\text{CheckDUnaware}(x_1^\circ, X1)$

is made. By this, the probability for CheckDUnaware$(x_1^\circ, X1)$ to abort due to $(1, \{(x_1, y_1), (x_1', y_1')\})$ is at most $\frac{2}{N-|EQueries|}$. Since there are at most 4μ such tuples (by Lemma 17), the total bound is $\frac{8\mu}{N-|EQueries|}$. Similar analysis establishes the following bounds:

- The probability of CheckDUnaware$(y_3^\circ, Y3)$ aborting does not exceed $\frac{8\mu}{N-|EQueries|}$ either;
- The probability of CheckDUnaware$(x_2, X2)$ aborting due to $(1, \{(x_1, y_1), (x_1', y_1')\})$ equals $Pr[y_1 \oplus x_2 \in \mathcal{Z} \vee y_1 \oplus k \oplus k' \oplus x_2 \in \mathcal{Z}]$ (for $\mathbf{R}.\mathrm{P1}(x_1) \to y_1$). Thus in total it's $\frac{8q_h\mu}{N-|P_1|}$. Similarly, the probability of CheckDUnaware$(y_2, Y2)$ is at most $\frac{8q_h\mu}{N-|P_3|}$.

It remains to consider calls to CheckDUnaware$(y_1, Y1)$ and CheckDUnaware$(x_3, X3)$. Such calls only occur in long cycles. Thus we assume a long cycle due to D querying $\mathrm{P}i^\delta(z) \to z'$ ($(i,\delta) \in \{(1,-),(3,+)\}$), and make discussion for each type of new 1- and 3-queries that are to be involved in CheckDUnaware-calls:

Case 1: the 1-query $(1, x_1, y_1, \leftarrow)$. Such queries are necessarily due to D querying $\mathrm{P1}^{-1}(y_1)$. It's not hard to see the above analysis can be similarly carried for such 1-queries, leading to the bound $\frac{8q_h^2\mu}{N-|P_1|}$. For a 3-query $(3, x_3, y_3, \rightarrow)$ we similarly obtain $\frac{8q_h^2\mu}{N-|P_3|}$.

Case 2: the 1-query $(1, x_1, y_1, \perp)$ *Created in* Process23TP. From the code of Process23TP and the analysis in Proposition 22, we find the fact that although G_2 queries $\mathbf{E}$ for x_1 to create this query, the value y_1 at the other side is computed without any additional randomness. Thus based on the queries not in *DUShoots* (note that this already includes all the earlier query-answer pairs obtained by D) and the last query $\mathrm{P}i^\delta(z)$ from D, the value y_1 can be fully determined, and does not increase the "knowledge" of D. This also means based on these values, the queries in *DUShoots* remain fully undermined, and distribute uniformly. Thus $Pr[$CheckDUnaware$(y_1, Y1)$ aborts$] \leq Pr[\exists(1, \{(x_1', y_1'), (x_1'', y_1'')\}) \in DUShoots : y_1' \oplus y_1 \in 2\mathcal{Z} \vee y_1'' \oplus y_1 \in 2\mathcal{Z}] \leq \frac{8q_h^2\mu}{N-|P_1|}$.

Case 3: the 1-query $(1, x_1, y_1, \perp)$ *Created in* ProcessShoot. In this case, there necessarily exists another 1-query $(1, x_1^*, y_1^*)$ and $k^*, k^{**} \in \mathcal{Z}$ such that G_2 obtains $y_1 \leftarrow y_1^* \oplus k^* \oplus k^{**}$ in this ProcessShoot-call. We further distinguish two cases:

(i) Right after CheckDUnaware(z, tag) (for the proper tag) returns, $(1, x_1^*, y_1^*)$ is not in *DUShoots*. Then the case is similar to *Case 2*: based on the queries not in *DUShoots* and the last query $\mathrm{P}i^\delta(z)$ from D, the value y_1 can be fully determined, while the queries in *DUShoots* remain undermined, and thus $Pr[$CheckDUnaware$(y_1, Y1)$ aborts$] \leq \frac{8q_h^2\mu}{N-|P_1|}$;

(ii) Right after CheckDUnaware(z, tag) returns, $(1, x_1^*, y_1^*)$ is in *DUShoots*. Then it necessarily be that G_2 "reaches" a shoot $(1, \{(x_1^*, y_1^*), (x_1^{**}, y_1^{**})\})$ in *DUShoots* when evaluating along the old E-chain (and soon remove this shoot). It's ensured that $y_1^* \oplus y_1 \in 2\mathcal{Z}$; however, when G_2 is to create $(1, x_1, y_1)$, $(1, \{(x_1^*, y_1^*), (x_1^{**}, y_1^{**})\})$ has been removed, and will not cause CheckDUnaware$(y_1, Y1)$ abort. Based on the additional values in this shoot, the other queries in *DUShoots* remain undermined, thus it holds $Pr[$CheckDUnaware$(y_1, Y1)$ aborts$] \leq \frac{8q_h^2\mu}{N-|P_1|}$.

Finally, by Lemma 17 we have $\frac{8q_h^2\mu}{N-|P_1|} \leq \frac{8q_h^2\mu}{N-|EQueries|}$. On the other hand, if $q_h \geq 1$ then $\frac{8\mu}{N-|EQueries|} \leq \frac{8q_h^2\mu}{N-|EQueries|}$; while when $q_h = 0$ we have $|DUShoots| = 0$ and CheckDUnaware never aborts. Thus the claim. □

Then the total bound for CheckDUnaware.

Lemma 19. *In D^{G_2}, the probability of* CheckDUnaware*-calls cause abort is at most* $\frac{32q_h^2\cdot(q_e+q_p)^2\cdot q_p^3}{N}$ *in total.*

Proof. We show that the number of CheckDUnaware-calls is at most $2(q_e + q_p) \cdot q_p$. This multiplied by the bound $\frac{8q_h^2\mu}{N-q_e-q_p-16\mu}$ given by Proposition 26 yields the claim (assuming $q_e + q_p + 16(q_e + q_p) \cdot q_p^2 \ll N/2$).

First, in each simulator cycle induced by D querying E, E^{-1}, P1, P2, $\mathrm{P2}^{-1}$, or $\mathrm{P3}^{-1}$, there's exactly one call to CheckDUnaware.

On the other hand, in a long cycle induced by D querying $\mathrm{P}i^{\delta}(z) \to z'$ ($(i,\delta) \in \{(1,-),(3,+)\}$), it can be seen from the code that G_2 would make a call to CheckDUnaware$(y_1, Y1)$ for each newly created 1-query $(1, x_1, y_1, d_1)$ such that $d_1 \in \{\leftarrow, \perp\}$, $x_1 \in EB(z')$, and DAwareness$(x_1, X1) = 1$ (and a call to CheckDUnaware$(x_3, X3)$ for each new 3-query $(3, x_3, y_3, d_3)$ such that $d_3 \in \{\to, \perp\}$, $y_3 \in EB(z')$, and DAwareness$(y_3, Y3) = 1$).[24] According to the analysis in *sub-case 4.1* of the proof of Proposition 13, we know the number of D-aware E-queries in $EB(z')$ does not exceed the total number of earlier E- and P-cycles, which is at most $q_e + q_p - 1$ (the current cycle excluded). By Lemma 9 we know these E-queries form a connected component (a sub-graph of $EB(z')$), thus these $q_e + q_p - 1$ E-queries provide $q_e + q_p$ nodes with DAwareness function value 1. Thus in each long cycle, there are at most $q_e + q_p$ DAwareness-calls.

By the above, the number of DAwareness-calls in total is at most $q_e + q_{p_2} + q_{p_1}(q_e + q_p) \le (q_p + 1)(q_e + q_p) \le 2(q_e + q_p) \cdot q_p$ (when $q_p \ge 1$). When $q_p = 0$ we have $|DUShoots| = 0$ and CheckDUnaware never aborts, thus the bound still holds. Thus the claim. ◻

11 From G_2 to the Final Indistinguishability Results

11.1 G_1 and G_2 Behave the same: Around Check Procedures

This subsection gives the transition from G_1 to G_2. The central issue is the procedure Check, and the argument just follows the idea initiated by Coron et al. [CPS08,HKT11] with no novelty. We thus omit the boring details, and directly apply the conclusion of [GL15b], which states that D's advantage in distinguishing G_1 and G_2 does not exceed twice the number of distinct calls to Check in D^{G_2} divided by N (a similar argument could be found in [DSSL16]).

Lemma 20. *(i)* $|Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} = 1] - Pr_{\mathbf{E},\mathbf{R}}[D^{G_1(\mathbf{E},\mathbf{R})} = 1]| \le \frac{338 q_h (q_e+q_p)^2 \cdot q_p^4}{N}$;

(ii) during the execution $D^{G_1(\mathbf{E},\mathbf{R})}$, with probability at least $1 - \frac{338 q_h (q_e+q_p)^2 \cdot q_p^4}{N}$, S issues at most $26 q_h (q_e + q_p) \cdot q_p^2$ queries to $\mathbf{E}$*, and runs in time at most $O((q_e + q_p)^2 \cdot q_p^4 + q_h (q_e + q_p)^2 \cdot q_p^4)$.*

Proof. The core idea lies in the fact that if the return values of all the calls to Check in D^{G_1} and D^{G_2} are the same, then the transcripts of queries and answers of D^{G_1} and D^{G_2} are the same, and D^{G_1} and D^{G_2} have the same running process. By Lemma 17, there are at most $169 q_h \cdot q_p^4 (q_e + q_p)^2$ Check-calls, thus the distinguishing bound $\epsilon \le 338 q_h \cdot q_p^4 (q_e + q_p)^2$ (assuming $\mathbf{E}$ is queried $q^* \ll N/2$ times in D^{G_1}). On the other hand, the distinguishing bound ϵ implies that the complexity of S in G_1 is consistent with the bounds given in Lemma 17, except with probability at most ϵ, thus the second claim. For the formal proof please see [GL15b].

The most time consuming procedure of S is ProcessShoot, with the four phases *Make-E-Chain*, *Shoot-Growing*, *Fill-in-Rung*, and *Shoot-Completing* requiring $O(q_e+q_p)$, $O(q_h(q_e+q_p) \cdot q_p^2)$, $O(q_e+q_p)$, and $O((q_e+q_p)^2 \cdot q_p^2)$ time respectively – note that the running time of *Shoot-Growing-Phase* is dominated by $O(q_h \mu)$ Check-calls, while that of *Shoot-Completing-Phase* is dominated by $O(q_e + q_p)$ CollectTP-calls, each can be implemented to run in time $O(\mu)$. As the number of ProcessShoot-calls is $O(q_p^2)$ (Lemma 16), ProcessShoot-calls cost $O(q_h(q_e + q_p) \cdot q_p^4 + (q_e + q_p)^2 \cdot q_p^4)$ in total. Meanwhile, we got $O(q_h(q_e + q_p)^2 \cdot q_p^4)$ calls to Check. Thus the running time in total is $O((q_e + q_p)^2 \cdot q_p^4 + q_h (q_e + q_p)^2 \cdot q_p^4)$. (The first term cannot be omitted, as the time cost cannot be 0 even if $q_h = 0$.) ◻

11.2 G_2 and G_3 Behave the same: the Partial Randomness Mapping

Consider the set *EQueries* standing at the end of a good G_2 execution. Note that E-queries (K, x_1, y_3) in *EQueries* can be divided into two types:

(i) **Type I**: (K, x_1, y_3) has been in a completed path;
(ii) **Type II**: (K, x_1, y_3) has not been in any completed paths.

Assume that the number of the two types are q_1 and q_2 (so $q_1 + q_2 = |EQueries|$) respectively. We denote by $\mathcal{EH}_2$ the set of **type II** E-queries, and denote by ST the tuple composed of *HQueries* and $\{P\}$, say, $ST = (HQueries, Queries)$. Finally, let $R = (\mathcal{EH}_2, ST)$.

[24] The new 1-/3-query may be created in RandAssign, in ProcessShoot, or in Process21/23TP, but this does not matter.

Then, the formalism of the randomness mapping part is very similar to [CS15b]. For this part, with respect to D, recall the notion *good* G_2*-tuple*: $\alpha = (\mathbf{E}, \mathbf{R})$ is a good G_2-tuple if the execution $D^{G_2(\alpha)}$ does not abort. Second, denote by $\mathcal{R}$ the set of all possible tuples of sets $R = (\mathcal{EH}_2, ST)$ standing at the end of good G_2 executions. For a good G_2-tuple $\alpha = (\mathbf{E}, \mathbf{R})$ and a tuple of sets $R \in \mathcal{R}$, if the sets $\mathcal{EH}_2$ and ST standing at the end of $D^{G_2(\alpha)}$ define exactly the same values as R (i.e. if R' are the sets of $D^{G_2(\alpha)}$ and $\forall (i, k, z), P_i(z) = P'_i(z)$ and $\forall K, HTable(K) = HTable(K')$ and the $\mathcal{EH}_2$ parts of the two set-tuples also contain the same contents), then write $D^{G_2(\alpha)} \to R$. Third, consider a set-tuple $R = (\mathcal{EH}_2, ST) \in \mathcal{R}$ with $ST = (HQueries, Queries)$. For a tuple of random primitive $\mathbf{R} = (\mathbf{H}, \mathbf{P})$, if for any $K \in HTable$ it holds $\mathbf{R}.\mathrm{H}(K) = HTable(K)$ and for any (i, z) such that $z \in E_i[k]$ it holds $\mathbf{R}.\mathrm{P}i(z) = P_i(z)$ (note this implies that for any $z' \in P_i^{-1}$ it also holds $\mathbf{R}.\mathrm{P}i^{-1}(z') = P_i^{-1}(z')$), then $\mathbf{R}$ *coincides with* ST, and denoted $\mathbf{R} \cong ST$.

We start with the following claim: the number of adapted (P-)queries (queries with $dir = \perp$) equals the number of **type I** E-queries. This slightly deviates from the previous works, which usually proved adapted queries as many as the targeted ideal-primitive-queries, but the idea is quite similar.

Lemma 21. *At the end of any good* G_2 *execution* $D^{G_2(\mathbf{E},\mathbf{R})}$*, it holds*

$$|\{(i, k, z, z', dir) \in Queries : dir = \perp\}| = |\mathit{Type\ I\ E\text{-}queries}|.$$

Proof. Note that right before each call to UpdateCompleted, there is a call to Adapt. Thus there is a bijective mapping between the completed paths and the AD-queries. As **type I** E-queries are in completed paths, this bijective mapping extends to **type I** E-queries and thus the claim. ◻

Following the spirit of H-coefficient technique, we should prove that the good-execution-history R has close probability to occur in G_2 and G_3 executions.

For the G_2 executions we consider the probability that it exactly generates the history R. We denote this value by $Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} \to R]$ and $Pr[D^{G_2} \to R]$ for short. Let $|HQueries| = h$ and $|EQueries| = w$, and assume that the number of 1-, 2-, and 3-queries created by RandAssign are r_1, r_2, and r_3 respectively. Then by Lemma 21, it should be $r_1 + r_2 + r_3 + q_1 = |P_1| + |P_2| + |P_3|$; and, obviously,

$$Pr[D^{G_2} \to R] \le \left(\prod_{i=1,2,3} \prod_{j=0}^{r_i - 1} \frac{1}{N - j} \right) \cdot \left(\frac{1}{N - q_1 - q_2} \right)^{q_1 + q_2} \cdot \frac{1}{N^h}.$$

We notice another probability:

$$Pr[\mathbf{R} \cong ST] = \left(\prod_{i=1,2,3} \prod_{j=0}^{|P_i| - 1} \frac{1}{N - j} \right) \cdot \frac{1}{N^h} \ge \left(\prod_{i=1,2,3} \prod_{j=0}^{r_i - 1} \frac{1}{N - j} \right) \cdot \frac{1}{N^{q_1}} \cdot \frac{1}{N^h}.$$

Moreover, it can be seen that when running on such $\mathbf{R} \cong ST$, D would get the same answer for all its H- and P- and **type I** E-queries in D^{G_2} and $D^{G_3(\mathbf{R})}$. Thus we should consider the probability that D's **type II** E-queries in D^{G_3} also lead to the same answers as in D^{G_2}. We denote this event by $\mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2$. To derive its probability, we first list several properties of **type II** E-queries.

Proposition 27. *For any* $R \in \mathcal{R}$*, let* $R = (\mathcal{EH}_2, ST)$ *and* $ST = (HQueries, Queries)$*. Then for any query* $(K, x_1, y_3) \in \mathcal{EH}_2$*, at least two among its six corresponding "round values" have not been defined by* ST*. More formally,* (K, x_1, y_3) *necessarily satisfy the following conditions:*

- *if* $x_1 \in P_1$ *then either* $y_3 \notin P_3^{-1}$ *or* $K \notin HTable$*. Furthermore, if* $K \in HTable$ *and* $k = HTable(K)$ *then:*
  - $P_1(x_1) \oplus k \notin P_2$*;*
  - *for any* $k' \in \mathcal{Z} \backslash \{k\}$*,* $P_1(x_1) \oplus k \oplus k' \notin P_1^{-1}$*;*
- *if* $y_3 \in P_3^{-1}$ *then either* $x_1 \notin P_1$ *or* $K \notin HTable$*. Furthermore, if* $K \in HTable$ *and* $k = HTable(K)$ *then:*
  - $P_3^{-1}(y_3) \oplus k \notin P_2^{-1}$*;*
  - *for any* $k' \in \mathcal{Z} \backslash \{k\}$*,* $P_3^{-1}(y_3) \oplus k \oplus k' \notin P_3$*.*

Proof. Consider the case of $x_1 \in P_1$, and assume the involved 1-query is $(1, x_1, y_1)$; the argument for the other case is similar by symmetry. First, if $y_3 \in P_3^{-1}$ and $K \in HTable$ simultaneously hold, then $(1, K, x_1)$ should have been in *Completed* by Inv6, and (K, x_1, y_3) should not have been a **type II** E-query. This shows either $y_3 \notin P_3^{-1}$ or $K \notin HTable$.

We then consider the case of $k = HTable(K)$. Under this condition, $P_1(x_1) \oplus k \notin P_2$ is obvious, as otherwise (K, x_1, y_3) should have been in a completed path by Inv6 and thus should not be **type II**. On the other hand, if there exists $k' \in \mathcal{Z} : k' \neq k$ such that $P_1(x_1) \oplus k \oplus k' \notin P_1^{-1}$, then the involved 1-query $(1, x'_1, y'_1, d'_1, n'_1)$ along with $(1, x_1, y_1, d_1, n_1)$ $(y'_1 = y_1 \oplus k \oplus k')$ form a 11-shoot. Now:

– If $x_1 \notin Border$, then (K, x_1, y_3) should have been in a completed path by Inv8 and Inv7;
– If $x_1 \in Border$, then DAWARENESS$(x_1, X1) = 1$ by Lemma 11. In this case, if (K, x_1, y_3) was "internally" created by G_2, then it should have been in a completed path by Lemma 3, a contradiction; if (K, x_1, y_3) was created due to D querying $\mathrm{E}/\mathrm{E}^{-1}$, then it would have caused G_2 abort in CHECKDUNAWARE.

Thus the claim. ◻

As the second step, with respect to given $\mathcal{EH}_2$, $Pr_{\mathbf{R}}[\mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2 \mid \mathbf{R} \cong ST]$ is easier to compute for the tuples meeting certain constraints. We call these tuples *good G_3-tuples* and *good* for short (here the approach is somewhat similar to [CS15a]).

First, for the set $\mathcal{EH}_2$, we extract the main keys as an extended key set $KSet$. Clearly $|KSet| \leq q_2$. Then we specify the first group of "bad conditions".

Definition 5. *With respect to $\mathcal{EH}_2$, $\mathbf{R}$.H is* bad*, if one of the following conditions is fulfilled:*

– *(BH-1) $\exists K \neq K' \in (KSet \cup HTable) : \mathbf{R}.\mathrm{H}(K) = \mathbf{R}.\mathrm{H}(K')$;*
– *(BH-2) $\exists K \in KSet \setminus HTable$ and $y_1 \in P_1^{-1} : y_1 \oplus \mathbf{R}.\mathrm{H}(K) \in P_2$;*
– *(BH-3) $\exists K \in KSet \setminus HTable$ and $x_3 \in P_3 : x_3 \oplus \mathbf{R}.\mathrm{H}(K) \in P_3^{-1}$;*
– *(BH-4) $\exists K \in KSet \setminus HTable$ and $K' \in (KSet \cup HTable)$ and $y_1 \neq y_1' \in P_1^{-1} : K \neq K'$ and $y_1 \oplus \mathbf{R}.\mathrm{H}(K) = y_1' \oplus \mathbf{R}.\mathrm{H}(K')$;*
– *(BH-5) $\exists K \in KSet \setminus HTable$ and $K' \in (KSet \cup HTable)$ and $x_3 \neq x_3' \in P_3 : K \neq K'$ and $x_3 \oplus \mathbf{R}.\mathrm{H}(K) = x_3' \oplus \mathbf{R}.\mathrm{H}(K')$;*

Under the conditions $\mathbf{R} \cong ST$ and $\mathbf{R}$.H is good, we further define *good* $\mathbf{R}$.P as follows.

Definition 6. *With respect to $\mathcal{EH}_2$, $\mathbf{R}$.P is* bad*, if one of the following conditions is fulfilled:*

– *(BP1-1) $\exists (K, x_1, y_3) \in \mathcal{EH}_2 : x_1 \notin P_1$ and $\mathbf{R}.\mathrm{P1}(x_1) \oplus \mathbf{R}.\mathrm{H}(K) \in P_2$;*
– *(BP1-2) $\exists (K, x_1, y_3) \neq (K', x_1', y_3') \in \mathcal{EH}_2 : x_1 \notin P_1$ and $\mathbf{R}.\mathrm{P1}(x_1) \oplus \mathbf{R}.\mathrm{H}(K) = \mathbf{R}.\mathrm{P1}(x_1') \oplus \mathbf{R}.\mathrm{H}(K')$;*
– *(BP3-1) $\exists (K, x_1, y_3) \in \mathcal{EH}_2 : y_3 \notin P_3^{-1}$ and $\mathbf{R}.\mathrm{P3}^{-1}(y_3) \oplus \mathbf{R}.\mathrm{H}(K) \in P_2^{-1}$;*
– *(BP3-2) $\exists (K, x_1, y_3) \neq (K', x_1', y_3') \in \mathcal{EH}_2 : y_3 \notin P_3^{-1}$ and $\mathbf{R}.\mathrm{P3}^{-1}(y_3) \oplus \mathbf{R}.\mathrm{H}(K) = \mathbf{R}.\mathrm{P3}^{-1}(y_3') \oplus \mathbf{R}.\mathrm{H}(K')$;*

Lemma 22. *For any $R = (\mathcal{EH}_2, ST) \in \mathcal{R}$, it holds*

$$Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad } \wedge \mathbf{R} \cong ST] \leq \frac{(q_2 + q_h) \cdot q_2 + 2q_2 \cdot |P_1| \cdot |P_2| + 2(q_2 + q_h) \cdot q_2 \cdot |P_1|^2}{N} + \frac{2q_2 \cdot |P_2| + 2q_2^2}{N - |P_1| - q_2}.$$

Proof. We first bound the probability of each condition corresponding to *bad* $\mathbf{R}$.H.

Condition (BH-1). Note that if $K, K' \in HTable$ then $\mathbf{R}.\mathrm{H}(K) = HTable(K) \neq HTable(K') = \mathbf{R}.\mathrm{H}(K')$, as we assumed $\mathbf{R} \cong ST$. Thus wlog assuming $K \notin HTable$. Then conditioned on $\mathbf{R} \cong ST$, $\mathbf{R}.\mathrm{H}(K)$ is an unknown random value, and thus $Pr[\mathbf{R}.\mathrm{H}(K) = \mathbf{R}.\mathrm{H}(K')] \leq 1/N$. The number of such key-pairs is at most $|KSet| \cdot (|HQueries| + |KSet|) \leq q_2(q_2 + q_h)$, thus $Pr[BH\text{-}1] \leq \frac{(q_2+q_h)\cdot q_2}{N}$.

The Others. Following the same line we got $Pr[BH\text{-}2] \leq \frac{|KSet|\cdot|P_1|\cdot|P_2|}{N}$, $Pr[BH\text{-}3] \leq \frac{|KSet|\cdot|P_2|\cdot|P_3|}{N}$, $Pr[BH\text{-}4] \leq \frac{|KSet|\cdot(|KSet|+|HQueries|)\cdot|P_1|^2}{N}$, and $Pr[BH\text{-}5] \leq \frac{|KSet|\cdot(|KSet|+|HQueries|)\cdot|P_3|^2}{N}$.

Assuming $\mathbf{R}$.H *good*, we then bound the probability of each condition corresponding to *bad* $\mathbf{R}$.P.

Condition (BP1-1), (BP3-1). If $x_1 \notin P_1$, then conditioned on $\mathbf{R} \cong ST$, $\mathbf{R}.\mathrm{P1}(x_1)$ can be seen as randomly picked from a pool of size at least $N - |P_1| - |\mathcal{EH}_2|$, thus for any such **type II** E-query (K, x_1, y_3) we have $Pr[\mathbf{R}.\mathrm{P1}(x_1) \oplus \mathbf{R}.\mathrm{H}(K) \in P_2] \leq |P_2|/(N - |P_1| - |\mathcal{EH}_2|)$, and in total $Pr[BP1\text{-}1] \leq \frac{q_2\cdot|P_2|}{N-|P_1|-q_2}$. For *(BP3-1)* the argument is similar by symmetry, resulting in $Pr[BP3\text{-}1] \leq \frac{q_2\cdot|P_2|}{N-|P_3|-q_2}$.

Condition (BP1-2), (BP3-2). Consider any two such **type II** E-queries (K, x_1, y_3) and (K', x_1', y_3'). We distinguish two cases as follows:

(i) $x_1 \neq x_1'$: then similarly to (*BP1-1*), it holds $Pr[\mathbf{R}.\text{P1}(x_1) = \mathbf{R}.\text{P1}(x_1') \oplus \mathbf{R}.\text{H}(K) \oplus \mathbf{R}.\text{H}(K')] \leq 1/(N - |P_1| - |\mathcal{EH}_2|)$.

(ii) $x_1 = x_1'$: then it necessarily be $K \neq K'$, and thus $\mathbf{R}.\text{H}(K) \neq \mathbf{R}.\text{H}(K')$ by $\neg$*BH-1*. Thus in this case, it must be $\mathbf{R}.\text{P1}(x_1) \oplus \mathbf{R}.\text{H}(K) \neq \mathbf{R}.\text{P1}(x_1') \oplus \mathbf{R}.\text{H}(K')$.

Thus for each pair of **type II** E-queries, the probability of (BP1-2) is at most $1/(N - |P_1| - |\mathcal{EH}_2|)$. As there are at most q_2^2 such pairs of E-queries, we got $Pr[BP1\text{-}2] \leq \frac{q_2^2}{N-|P_1|-q_2}$. For (*BP3-1*) the analysis is similar by symmetry, obtaining $Pr[BP3\text{-}1] \leq \frac{q_2^2}{N-|P_3|-q_2}$. The above sum up to the bound (note that the upper bounds of $|P_1|$ and $|P_3|$ are equal). $\square$

Finally, we are able to use the following lemma to bound the probability of $\mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2$ conditioned on $\mathbf{R} \cong ST$. The core idea is to show that each such **type II** E-query would lead to a new (random) pair of input and output of $\mathbf{P}_2$.

Lemma 23. *$Pr_{\mathbf{R}}[EMR_3^*(\mathbf{R}) \cong \mathcal{EH}_2 \mid \mathbf{R} \cong ST] \geq (1 - Pr[\mathbf{R}$ is bad$]) \cdot \frac{1}{N^{q_2}}$.*

Proof. Let $ST = (HQueries, Queries)$. Consider the $(l+1)^{\text{th}}$ **type II** E-query $(K^{l+1}, x_1^{l+1}, y_3^{l+1})$. Conditioned on $\mathbf{R} \cong ST$ and $\mathbf{R}$ is good, we show that each $(K^{l+1}, x_1^{l+1}, y_3^{l+1})$ can be associated with a unique pair (x_2^{l+1}, y_2^{l+1}) such that $x_2^{l+1} \notin P_2$ and $y_2^{l+1} \notin P_2^{-1}$, so that $Pr_{\mathbf{R}}[\mathsf{EMR}_3^*(\mathbf{R})(K^{l+1}, x_1^{l+1}) \to y_3^{l+1}] = Pr_{\mathbf{R}}[\mathbf{R}.\text{P2}(x_2^{l+1}) = y_2^{l+1}] \geq \frac{1}{N}$. More clearly, for $(K^{l+1}, x_1^{l+1}, y_3^{l+1})$, we let $k^{l+1} = \mathbf{R}.\text{H}(K^{l+1})$, $x_2^{l+1} = \mathbf{R}.\text{P1}(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = \mathbf{R}.\text{P3}^{-1}(y_3^{l+1}) \oplus k^{l+1}$. We note that **type II** E-queries can be grouped into the following seven (somewhat similar to [CLS15]. See Fig. 12 for illustration.):

- $Group_1 = \{(K, x_1, y_3) \in \mathcal{EH}_2 : K \in HTable \text{ and } x_1 \in P_1\}$.
  - If $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_1$ then $k^{l+1} = HTable(K^{l+1})$, $x_2^{l+1} = P_1(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = \mathbf{R}.\text{P3}^{-1}(y_3^{l+1}) \oplus k^{l+1}$. Meanwhile, $x_2^{l+1} \notin P_2$ by Proposition 27, while $y_2^{l+1} \notin P_2^{-1}$ since $\mathbf{R}.\text{P}$ is good (more clearly, since *BP3-1* does not occur);
- $Group_2 = \{(K, x_1, y_3) \in \mathcal{EH}_2 : K \in HTable \text{ and } y_3 \in P_3^{-1}\}$.
  - If $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_2$ then $k^{l+1} = HTable(K^{l+1})$, $x_2^{l+1} = \mathbf{R}.\text{P1}(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = P_3^{-1}(y_3^{l+1}) \oplus k^{l+1}$. And $y_2^{l+1} \notin P_2^{-1}$ by Proposition 27, while $x_2^{l+1} \notin P_2$ follows from the goodness of $\mathbf{R}.\text{P}$ (more clearly, $\neg$*BP1-1*);
- $Group_3 = \{(K, x_1, y_3) \in \mathcal{EH}_2 : K \notin HTable \text{ and } x_1 \in P_1 \text{ and } y_3 \in P_3^{-1}\}$.
  - If $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_3$ then $k^{l+1} = \mathbf{R}.\text{H}(K^{l+1})$, $x_2^{l+1} = P_1(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = P_3^{-1}(y_3^{l+1}) \oplus k^{l+1}$. Now $x_2^{l+1} \notin P_2$ and $y_2^{l+1} \notin P_2^{-1}$ follow from $\neg$*BH-2* and $\neg$*BH-3* respectively;
- $Group_4 = \{(K, x_1, y_3) \in \mathcal{EH}_2 : K \notin HTable \text{ and } x_1 \in P_1 \text{ and } y_3 \notin P_3^{-1}\}$.
  - If $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_4$ then it holds $k^{l+1} = \mathbf{R}.\text{H}(K^{l+1})$, $x_2^{l+1} = P_1(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = \mathbf{R}.\text{P3}^{-1}(y_3^{l+1}) \oplus k^{l+1}$. Now $x_2^{l+1} \notin P_2$ by $\neg$*BH-2*, and $y_2^{l+1} \notin P_2^{-1}$ by $\neg$*BP3-1*;
- $Group_5 = \{(K, x_1, y_3) \in \mathcal{EH}_2 : K \notin HTable \text{ and } x_1 \notin P_1 \text{ and } y_3 \in P_3^{-1}\}$.
  - If $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_5$ then $k^{l+1} = \mathbf{R}.\text{H}(K^{l+1})$, $x_2^{l+1} = \mathbf{R}.\text{P1}(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = P_3^{-1}(y_3^{l+1}) \oplus k^{l+1}$. Now $x_2^{l+1} \notin P_2$ by $\neg$*BP1-1*, and $y_2^{l+1} \notin P_2^{-1}$ by $\neg$*BH-3*;
- $Group_6 = \{(K, x_1, y_3) \in \mathcal{EH}_2 : K \in HTable \text{ and } x_1 \notin P_1 \text{ and } y_3 \notin P_3^{-1}\}$.
  - If $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_6$ then $k^{l+1} = HTable(K^{l+1})$, $x_2^{l+1} = \mathbf{R}.\text{P1}(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = \mathbf{R}.\text{P3}^{-1}(y_3^{l+1}) \oplus k^{l+1}$. Now $x_2^{l+1} \notin P_2$ and $y_2^{l+1} \notin P_2^{-1}$ follow from $\neg$*BP1-1* and $\neg$*BP3-1* respectively.
- $Group_7 = \{(K, x_1, y_3) \in \mathcal{EH}_2 : K \notin HTable \text{ and } x_1 \notin P_1 \text{ and } y_3 \notin P_3^{-1}\}$.
  - If $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_7$ then $k^{l+1} = \mathbf{R}.\text{H}(K^{l+1})$, $x_2^{l+1} = \mathbf{R}.\text{P1}(x_1^{l+1}) \oplus k^{l+1}$, and $y_2^{l+1} = \mathbf{R}.\text{P3}^{-1}(y_3^{l+1}) \oplus k^{l+1}$. Similarly to $Group_5$, $x_2^{l+1} \notin P_2$ and $y_2^{l+1} \notin P_2^{-1}$ follow from $\neg$*BP1-1* and $\neg$*BP3-1* respectively.

We then show that the associated (x_2^{l+1}, y_2^{l+1}) would not collide with the associated (x_2^j, y_2^j) for some **type II** E-query (K^j, x_1^j, y_3^j) with $j < l+1$. For this we consider the following possibilities:

Case I: $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_1$. Depending on which group (K^j, x_1^j, y_3^j) belongs to, we got seven possibilities. However, the key points can be summarized as follows:

(i) If $x_1^j \notin P_1$, then $x_2^j \neq x_2^{l+1}$ by $\neg$*BP1-2*. If $x_1^j \in P_1$, then if $K^j \notin HTable$, then $x_2^j \neq x_2^{l+1}$ by $\neg$*BH-4*, otherwise by Proposition 27;

(ii) On the other hand, in each case, $y_2^j \neq y_2^{l+1}$ holds by $\neg$*BP3-2*.

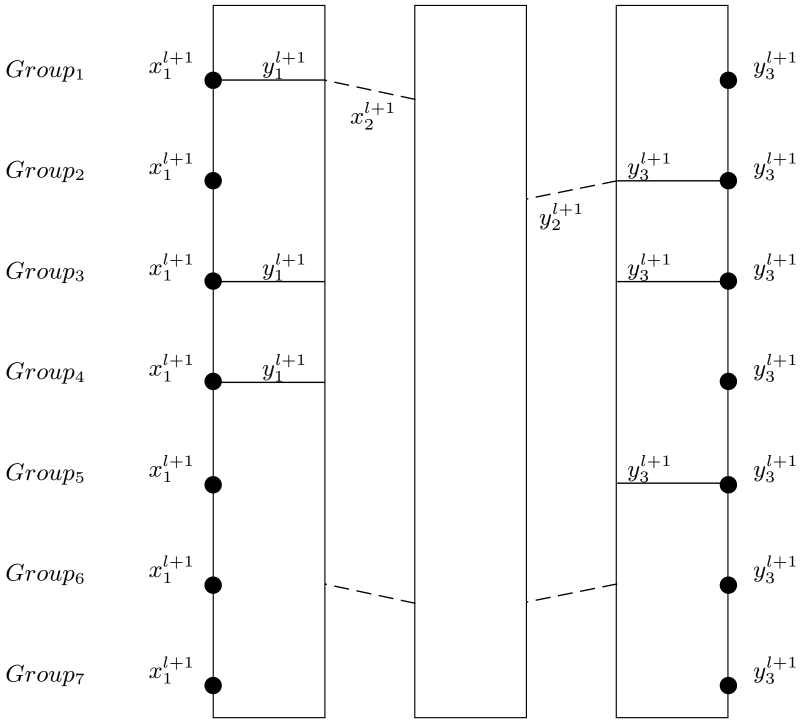


Fig. 12. Groups of **type II** E-queries.

Case II: $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_2$. Depending on which features (K^j, x_1^j, y_3^j) possesses, we have the following discussion (symmetrically to *Case I*):

(i) $x_2^j \neq x_2^{l+1}$ follows from $\neg$*BP1-2*;
(ii) If $y_3^j \notin P_3^{-1}$, then $y_2^j \neq y_2^{l+1}$ by $\neg$*BP3-2*. Otherwise, if $K^j \notin HTable$, then $y_2^j \neq y_2^{l+1}$ by $\neg$*BH-5*, else by Proposition 27;

Case III: $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_3$. Depending on which group (K^j, x_1^j, y_3^j) belongs to, we got four possibilities:

(i) $(K^j, x_1^j, y_3^j) \in Group_6 \cup Group_7$. Then we have $x_1^j \notin P_1$ and $y_3^j \notin P_3^{-1}$, and thus $x_2^j \neq x_2^{l+1}$ and $y_2^j \neq y_2^{l+1}$ follow from $\neg$*BP1-2* and $\neg$*BP3-2* respectively;
(ii) $(K^j, x_1^j, y_3^j) \in Group_5$. Then $x_2^j \neq x_2^{l+1}$ follows from $\neg$*BP1-2*. On the other hand, if $K^j \neq K^{l+1}$, then $y_2^j \neq y_2^{l+1}$ by $\neg$*BH-5*; otherwise, it necessarily holds $y_3^j \neq y_3^{l+1}$ and thus $y_2^j \neq y_2^{l+1}$ is guaranteed;
(iii) $(K^j, x_1^j, y_3^j) \in Group_4$. Similar to the previous case by symmetry, $y_2^j \neq y_2^{l+1}$ follows from $\neg$*BP3-2*, while $x_2^j \neq x_2^{l+1}$ follows from $\neg$*BH-4* when $K^j \neq K^{l+1}$ (and is ensured otherwise);
(iv) $(K^j, x_1^j, y_3^j) \in Group_3$. Then, if $K^j \neq K^{l+1}$, then $x_2^j \neq x_2^{l+1}$ follows from $\neg$*BH-4* while $y_2^j \neq y_2^{l+1}$ by $\neg$*BH-5*; otherwise, it has to be $x_1^j \neq x_1^{l+1} \Rightarrow x_2^j \neq x_2^{l+1}$ and $y_3^j \neq y_3^{l+1} \Rightarrow y_2^j \neq y_2^{l+1}$;
(v) $(K^j, x_1^j, y_3^j) \in Group_1 \cup Group_2$. These subcases have been taken into account in the above analysis of *Case I* and *II*.

Case IV: $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_4$. Depending on which group (K^j, x_1^j, y_3^j) belongs to, we got three possibilities:

(i) $(K^j, x_1^j, y_3^j) \in Group_5 \cup Group_6 \cup Group_7$. Then $x_1^j \notin P_1$ and $x_2^j \neq x_2^{l+1}$ by $\neg$*BP1-2* while $y_2^{l+1} \notin P_3^{-1}$ and thus $y_2^j \neq y_2^{l+1}$ by $\neg$*BP3-2*;
(ii) $(K^j, x_1^j, y_3^j) \in Group_4$. Then $y_2^j \neq y_2^{l+1}$ follows from $\neg$*BP3-2*. On the other hand, if $K^j \neq K^{l+1}$, then $x_2^j \neq x_2^{l+1}$ follows from $\neg$*BH-4*; otherwise we have $x_1^j \neq x_1^{l+1}$ and further $x_2^j \neq x_2^{l+1}$;
(iii) $(K^j, x_1^j, y_3^j) \in Group_1 \cup Group_2 \cup Group_3$. Already included in *Case I-III*.

Case V: $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_5$. Then we got three possibilities:

(i) $(K^j, x_1^j, y_3^j) \in Group_6 \cup Group_7$. Then $x_2^j \neq x_2^{l+1}$ by $\neg$*BP1-2*, while $y_2^j \neq y_2^{l+1}$ by $\neg$*BP3-2*;
(ii) $(K^j, x_1^j, y_3^j) \in Group_5$. Then $x_2^j \neq x_2^{l+1}$ follows from $\neg$*BP1-2*. On the other hand, if $K^j \neq K^{l+1}$, then $y_2^j \neq y_2^{l+1}$ follows from $\neg$*BH-5*; otherwise, it has to be $y_3^j \neq y_3^{l+1}$ and $y_2^j \neq y_2^{l+1}$ is ensured;
(iii) $(K^j, x_1^j, y_3^j) \in Group_1 \cup Group_2 \cup Group_3$. Already included in *Case I-IV*.

Case VI: $(K^{l+1}, x_1^{l+1}, y_3^{l+1}) \in Group_6 \cup Group_7$. Then:

(i) If $(K^j, x_1^j, y_3^j) \in Group_6 \cup Group_7$, then $x_2^j \neq x_2^{l+1}$ by $\neg$*BP1-2* and $y_2^j \neq y_2^{l+1}$ by $\neg$*BP3-2*;
(ii) The other cases have been included in *Case I-V*.

By the above, for each **type II** E-query $(K^{l+1}, x_1^{l+1}, y_3^{l+1})$ it indeed holds $Pr_{\mathbf{R}}[\mathsf{EMR}_3^*(\mathbf{R})(K^{l+1}, x_1^{l+1}) \to y_3^{l+1}] = Pr_{\mathbf{R}}[\mathbf{R}.\mathrm{P2}(x_2^{l+1})] \geq \frac{1}{N}$. Thus

$$\begin{aligned}&Pr_{\mathbf{R}}[\mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2 \mid \mathbf{R} \cong ST]\\ \geq& Pr[\mathbf{R} \text{ is good}] \cdot \prod_{l=0}^{q_2-1} Pr[\mathsf{EMR}_3^*(\mathbf{R})(K^{l+1}, x_1^{l+1}) \to y_3^{l+1} \mid (\mathbf{R} \cong ST \wedge \mathbf{R} \text{ is good})]\\ \geq&(1 - Pr[\mathbf{R} \text{ is bad}]) \cdot \frac{1}{N^{q_2}}\end{aligned}$$

as claimed. $\square$

Thus the ratio:

Lemma 24. *For any $R \in \mathcal{R}$, it holds*

$$\frac{Pr_{\mathbf{R}}[\mathit{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2 \wedge \mathbf{R} \cong ST]}{Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} \to R]} \geq 1 - \frac{w^2}{N} - Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad}].$$

Proof. By Lemma 23 and the above discussions we have:

$$\begin{aligned}&\frac{Pr_{\mathbf{R}}[\mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2 \wedge \mathbf{R} \cong ST]}{Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} \to R]} \geq \frac{Pr_{\mathbf{R}}[\mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2 \mid \mathbf{R} \cong ST] \cdot Pr_{\mathbf{R}}[\mathbf{R} \cong ST]}{Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} \to R]}\\ \geq& \frac{(1 - Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad}]) \cdot \frac{1}{N^{q_2}} \cdot \left(\prod_{i=1,2,3} \prod_{j=0}^{r_i-1} \frac{1}{N-j}\right) \cdot \frac{1}{N^{q_1}} \cdot \frac{1}{N^h}}{\left(\prod_{i=1,2,3} \prod_{j=0}^{r_i-1} \frac{1}{N-j}\right) \cdot (\frac{1}{N-q_1-q_2})^{q_1+q_2} \cdot \frac{1}{N^h}}\\ \geq&(1 - Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad}]) \cdot \left(\frac{N-w}{N}\right)^w \geq (1 - Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad}]) \cdot \left(1 - \frac{w^2}{N}\right) \geq 1 - \frac{w^2}{N} - Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad}]\end{aligned}$$

Thus the claim. $\square$

The above already exhibited a sufficient condition for D giving the same output during the interactions with G_2 and G_3.

Lemma 25. *For any good G_2-tuple $(\mathbf{E}, \mathbf{R})$ and $\mathbf{R}'$, if the following three are simultaneously fulfilled,*

- $D^{G_2(\mathbf{E},\mathbf{R})} \to R(= (\mathcal{EH}_2, ST))$;
- $\mathbf{R}' \cong ST$;
- $\mathit{EMR}_3^*(\mathbf{R}') \cong \mathcal{EH}_2$;

then the transcripts of queries and answers of D in the two executions $D^{G_2(\mathbf{E},\mathbf{R})}$ and $D^{G_3(\mathbf{R}')}$ are the same, and D gives the same output: $D^{G_2(\mathbf{E},\mathbf{R})} = D^{G_3(\mathbf{R}')}$.

Proof. We show the claim via an induction on D's transcript of queries and answers. Assume that the transcripts of D in the two executions are the same up to some point, and consider the next query. As D is deterministic, D's next queries in the two executions are the same. We prove that D obtains the same answer. For this we consider the following possibilities:

(i) the query is to H/Pi/Pi^{-1}: then the answers are the same, since the answer obtained in $D^{G_2(\mathbf{E},\mathbf{R})}$ equals the value in ST, and $\mathbf{R}' \cong ST$;

(ii) the query is an E-query, and it does not fall into $\mathcal{EH}_2$. This means this query turns out **type I** when the G_2 execution ends. Then as **type I** E-queries are in completed paths, and the values of the corresponding path are in ST which are followed by $\mathbf{R}'$, the answers obtained in $D^{G_2(\mathbf{E},\mathbf{R})}$ and $D^{G_3(\mathbf{R}')}$ are the same;

(iii) the query is an E-query which falls into $\mathcal{EH}_2$. Then as we assumed $\mathsf{EMR}_3^*(\mathbf{R}') \cong \mathcal{EH}_2$, the answer obtained in $D^{G_2(\mathbf{E},\mathbf{R})}$ and $D^{G_3(\mathbf{R}')}$ are the same.

Therefore, the answers are the same, and the two transcripts of D turns out the same as the induction proceeds. Since D is deterministic, D outputs the same in the two executions. □

Good G_2 executions can be partitioned with respect to the sets generated by them: for any $R \in \mathcal{R}$ and any two tuples $(\mathbf{E},\mathbf{R})$ and $(\mathbf{E}',\mathbf{R}')$, once $D^{G_2(\mathbf{E},\mathbf{R})} \to R$ and $D^{G_2(\mathbf{E}',\mathbf{R}')} \to R$, then $D^{G_2(\mathbf{E},\mathbf{R})} = D^{G_2(\mathbf{E}',\mathbf{R}')}$.

Lemma 26. $Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} = 1] = \sum_{R\in\mathcal{R}:\exists(\mathbf{E}^*,\mathbf{R}^*)\ s.t.\ D^{G_2(\mathbf{E}^*,\mathbf{R}^*)}\to R \wedge D^{G_2(\mathbf{E}^*,\mathbf{R}^*)}=1} Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} \to R]$.

Proof. We proceed to argue that for any $R = (\mathcal{EH}_2, ST) \in \mathcal{R}$, if there is a tuple $(\mathbf{E}^*,\mathbf{R}^*)$ such that $D^{G_2(\mathbf{E}^*,\mathbf{R}^*)} \to R$, then for any tuple $(\mathbf{E},\mathbf{R})$ such that $D^{G_2(\mathbf{E},\mathbf{R})} \to R$, it holds $D^{G_2(\mathbf{E},\mathbf{R})} = D^{G_2(\mathbf{E}^*,\mathbf{R}^*)}$. For this we show that the transcripts of queries and answers in the two executions $D^{G_2(\mathbf{E},\mathbf{R})}$ and $D^{G_2(\mathbf{E}^*,\mathbf{R}^*)}$ are the same. The transcripts encode all the randomness that influences the executions, thus they include queries to H, Pi, Pi^{-1}, E, E^{-1}, and Check. We use an induction similar to Lemma 25 – we assume the transcripts generated so far are the same and consider the next query:

(i) the query is to H/Pi/Pi^{-1}: then the answers are the same, since the answer equals the value in R, and both $D^{G_2(\mathbf{E},\mathbf{R})} \to R$ and $D^{G_2(\mathbf{E}^*,\mathbf{R}^*)} \to R$;

(ii) the query is an E-query, and it does not fall into $\mathcal{EH}_2$. This means in both $D^{G_2(\mathbf{E},\mathbf{R})}$ and $D^{G_2(\mathbf{E}^*,\mathbf{R}^*)}$, the query is **type I**. Then as **type I** E-queries are in completed paths, and the values of the corresponding path are in ST, and both $D^{G_2(\mathbf{E},\mathbf{R})} \to R$ and $D^{G_2(\mathbf{E}^*,\mathbf{R}^*)} \to R$, the answers obtained in $D^{G_2(\mathbf{E},\mathbf{R})}$ and $D^{G_2(\mathbf{E}^*,\mathbf{R}^*)}$ are the same;

(iii) the query is an E-query in $\mathcal{EH}_2$. Then the answers clearly equal, as both $D^{G_2(\mathbf{E},\mathbf{R})} \to (\mathcal{EH}_2, ST)$ and $D^{G_2(\mathbf{E}^*,\mathbf{R}^*)} \to (\mathcal{EH}_2, ST)$;

(iv) the query is to Check: as the transcripts obtained so far are equal, the entries in *EQueries* in the two executions are also the same, so that the answers to Check are the same.

Hence the transcripts obtained by D are also the same and thus $D^{G_2(\mathbf{E},\mathbf{R})} = D^{G_2(\mathbf{E}^*,\mathbf{R}^*)}$. These complete the proof. □

With Lemma 26 in mind, let Θ_1 be the subset of $\mathcal{R}$ such that for any tuple $(\mathbf{E},\mathbf{R})$ such that $D^{G_2(\mathbf{E},\mathbf{R})} \to R \in \Theta_1$ it holds $D^{G_2(\mathbf{E},\mathbf{R})} = 1$. Then we have the following inequality.

Lemma 27. $Pr_{\mathbf{R}}[D^{G_3(\mathbf{R})} = 1] \geq \sum_{R=(\mathcal{EH}_2,ST)\in\Theta_1} Pr_{\mathbf{R}}[\mathbf{R} \cong ST \wedge \mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2]$.

Proof. We show that for any tuple $\mathbf{R}^*$, there is at most one $R = (\mathcal{EH}_2, ST) \in \mathcal{R}$ s.t. $\mathbf{R}^* \cong ST \wedge \mathsf{EMR}_3^*(\mathbf{R}^*) \cong \mathcal{EH}_2$. Assume otherwise, i.e. $\exists R' = (\mathcal{EH}_2', ST') \in \mathcal{R}$ such that:

– $R \neq R'$;
– $\mathbf{R}^* \cong ST \wedge \mathbf{R}^* \cong ST'$;
– $\mathsf{EMR}_3^*(\mathbf{R}^*) \cong \mathcal{EH}_2'$ (not necessarily $\mathcal{EH}_2' = \mathcal{EH}_2$).

Assume that for two good tuples $\alpha = (\mathbf{E},\mathbf{R})$ and $\alpha' = (\mathbf{E}',\mathbf{R}')$, it holds $D^{G_2(\alpha)} \to R$ and $D^{G_2(\alpha')} \to R'$. Note that in $D^{G_2(\alpha)}$, for each **type I** E-query, the values in the corresponding chain are in ST, which are followed by $\mathbf{R}^*$. Meanwhile, $\mathsf{EMR}_3^*(\mathbf{R}^*) \cong \mathcal{EH}_2$. Thus for each E-query (K, x_1, y_3) appeared in $D^{G_2(\alpha)}$ it holds $y_3 = \mathsf{EMR}_3^*(\mathbf{R}^*).\mathrm{E}(K, x_1)$. Similar claim holds for $D^{G_2(\alpha')}$. By these observations and an induction similar to Lemma 26, we could show the transcripts (cf. Lemma 26) of the two executions $D^{G_2(\alpha)}$ and $D^{G_2(\alpha')}$ are the same, so that the two set-tuples R and R' should be the same, which is a contradiction. Assume the transcripts obtained so far are the same and consider the next query:

(i) the query is to H/Pi/Pi^{-1}: the answers are the same, as they equal the corresponding entries in ST and ST' respectively, and $\mathbf{R}^* \cong ST \wedge \mathbf{R}^* \cong ST'$;

(ii) the query is an E-query $\mathrm{E}(K, x_1)$. Then as argued, both of the two answers are equal to $\mathsf{EMR}_3^*(\mathbf{R}^*).\mathrm{E}(K, x_1)$ and thus the same. Similarly for decryption query $\mathrm{E}^{-1}(K, y_3)$;

(iii) the query is to CHECK: similarly to Lemma 26, the answers are the same.

The above establish that for any tuple $\mathbf{R}^*$, there exists at most one $R = (\mathcal{EH}_2, ST) \in \mathcal{R}$ s.t. $\mathbf{R}^* \cong ST$ and $\mathsf{EMR}_3^*(\mathbf{R}^*) \cong \mathcal{EH}_2$. After this, we have

$$\begin{aligned} Pr_{\mathbf{R}}[D^{G_3(\mathbf{R})} = 1] &\geq Pr_{\mathbf{R}}[D^{G_3(\mathbf{R})} = 1 \wedge \exists R = (\mathcal{EH}_2, ST) \in \mathcal{R} \text{ s.t. } \mathbf{R} \cong ST \wedge \mathsf{EMR}_3^*(\mathbf{R}^*) \cong \mathcal{EH}_2] \\ &= \sum_{R=(\mathcal{EH}_2,ST)\in\Theta_1} Pr_{\mathbf{R}}[\mathbf{R} \cong ST \wedge \mathsf{EMR}_3^*(\mathbf{R}^*) \cong \mathcal{EH}_2] \text{ (by Lemma 25)} \end{aligned}$$

as claimed. □

The above finally yields the following distinguishing bound.

Lemma 28. *The advantage of D distinguishing G_2 and G_3 is at most*

$$\begin{aligned} &|Pr_{\mathbf{R}}[D^{G_3(\mathbf{R})} = 1] - Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} = 1]| \\ \leq &\frac{2176 q_h^6 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{2359 q_e^2 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{338 q_e \cdot q_h (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{q_h^2 + q_h^4 + 8q_e^2 + q_e \cdot q_h}{N}. \end{aligned}$$

Proof. Wlog assume that $Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} = 1] \geq Pr_{\mathbf{R}}[D^{G_3(\mathbf{R})} = 1]$, then

$$\begin{aligned} &|Pr_{\mathbf{R}}[D^{G_3(\mathbf{R})} = 1] - Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} = 1]| \\ \leq &\underbrace{Pr_{\mathbf{E},\mathbf{R}}[(\mathbf{E},\mathbf{R}) \text{ is not a good } G_2\text{-tuple}]}_{\leq Pr_{\mathbf{E},\mathbf{P}}[D^{G_2(\mathbf{E},\mathbf{R})} \text{ aborts}]} \\ &+ Pr_{\mathbf{E},\mathbf{R}}[(\mathbf{E},\mathbf{R}) \text{ is a good } G_2\text{-tuple} \wedge D^{G_2(\mathbf{E},\mathbf{R})} = 1] - Pr_{\mathbf{E}}[D^{G_3(\mathbf{E})} = 1] \\ \leq &Pr_{\mathbf{E},\mathbf{P}}[D^{G_2(\mathbf{E},\mathbf{R})} \text{ aborts}] + \sum_{R\in\Theta_1} Pr_{\mathbf{E},\mathbf{P}}[D^{G_2(\mathbf{E},\mathbf{R})} \to R] \text{ (by Lemma 26)} \\ &- \sum_{R\in\Theta_1} Pr_{\mathbf{R}}[\mathbf{R} \cong ST \wedge \mathsf{EMR}_3^*(\mathbf{R}) \cong \mathcal{EH}_2] \text{ (by Lemma 27)} \\ \leq &Pr_{\mathbf{E},\mathbf{P}}[D^{G_2(\mathbf{E},\mathbf{R})} \text{ aborts}] + \sum_{R\in\Theta_1} \left(\frac{w^2}{N} + Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad}]\right) \cdot Pr_{\mathbf{E},\mathbf{R}}[D^{G_2(\mathbf{E},\mathbf{R})} \to R] \text{ (by Lemma 24)} \\ \leq &Pr_{\mathbf{E},\mathbf{P}}[D^{G_2(\mathbf{E},\mathbf{R})} \text{ aborts}] + \frac{w^2}{N} + Pr_{\mathbf{R}}[\mathbf{R} \text{ is bad}] \end{aligned}$$

Gathering the bounds given in Lemmata 18, 19, 22, and 17, and assuming $13(q_e + q_p) \cdot q_p^2 + q_e \ll N/2$, we obtain the following upper bound:

$$\begin{aligned} &\frac{(1462 + 2144 q_h^6) \cdot (q_e + q_p)^2 \cdot q_p^4 + 2q_e^2 + q_h^2 + q_h^4}{N} + \frac{32 q_h^2 (q_e + q_p)^2 \cdot q_p^3}{N} \\ &+ \left(\frac{(q_e + q_h) \cdot q_e + 234 q_e (q_e + q_p)^2 \cdot q_p^4 + 338 q_e (q_e + q_h)(q_e + q_p)^2 \cdot q_p^4}{N} + \frac{18 q_e (q_e + q_p) \cdot q_p^2 + 2q_e^2}{N - 13(q_e + q_p) \cdot q_p^2 - q_e}\right) \\ &+ \frac{(q_e + q_p + 16(q_e + q_p) \cdot q_p^2)^2 \ (\leq q_e^2 + 17^2 (q_e + q_p)^2 \cdot q_p^4)}{N} \\ \leq &\frac{2176 q_h^6 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{2359 q_e^2 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{338 q_e \cdot q_h (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{q_h^2 + q_h^4 + 8q_e^2 + q_e \cdot q_h}{N}. \end{aligned}$$

as claimed. □

12 To EMR_3: a Formal Proof

This section proves Theorem 1 based on Theorem 2. We first describe the simulator $\widetilde{S}^{\mathbf{E},\mathbf{R}}$. $\widetilde{S}^{\mathbf{E},\mathbf{R}}$ runs S (in Section 5), relaying S's queries to $\mathbf{R}$. On the other hand, each time S issues a query $\mathrm{E}^{\delta}(K, z)$, $\widetilde{S}$ sets $k \leftarrow HTable(K)$ and answers with $k \oplus \mathrm{E}^{\delta}(K, k \oplus z)$. This design requires $K \in HTable$ before S issuing such query; by checking the code, it's easy to verify that S indeed meets this constraint.

We prove that for this $\widetilde{S}$, it holds $\mathrm{Adv}^{indif}_{\mathsf{EMR}_3,\mathbf{E},\widetilde{S}} \leq \mathrm{Adv}^{indif}_{\mathsf{EMR}_3^*,\mathbf{E},S}$ for any distinguisher $\widetilde{D}$. To this end, we consider the following sequence of games:

$\widetilde{G}_1$: this game takes $(\mathbf{E}, \mathbf{R})$ as the randomness source, and captures the interaction between $\widetilde{D}$, $\widetilde{S}$, and $\mathbf{E}$. After $\widetilde{D}$ outputs, $\widetilde{G}_1$ outputs the same as $\widetilde{D}$.

$\widetilde{G}_2$: this imagined game takes $(\mathbf{E}, \mathbf{R})$ as the randomness source, and captures the interaction between $\widetilde{D}$, $\widetilde{S}$, and a "shelled" cipher $\widetilde{E}^{\mathbf{E},\mathbf{R}}$. Upon each query $\mathrm{E}^{\delta}(K, z)$, $\widetilde{E}^{\mathbf{E},\mathbf{R}}$ gets $k \leftarrow \mathbf{R}.\mathrm{H}(K)$ and answers with $k \oplus \mathrm{E}^{\delta}(K, k \oplus z)$. As an imagined intermediate game, such design is not problematic. $\widetilde{G}_2$ also outputs the same as $\widetilde{D}$. Clearly, for any random tuple $(\mathbf{E}, \mathbf{R})$, it's easy to find a corresponding tuple $(\mathbf{E}^*, \mathbf{R})$ such that $\widetilde{D}$ and $\widetilde{S}$ generate the same transcript of queries and answers in $\widetilde{G}_1(\mathbf{E}, \mathbf{R})$ and $\widetilde{G}_2(\mathbf{E}^*, \mathbf{R})$, thus $Pr[\widetilde{G}_1 = 1] = Pr[\widetilde{G}_2 = 1]$.

$\widetilde{G}_3$: this imagined game takes $(\mathbf{E}, \mathbf{R})$ as the randomness source, and captures the interaction between an "illegal" distinguisher D_{il}, the simulator $S^{\mathbf{E},\mathbf{R}}$, and the ideal cipher $\mathbf{E}$. The distinguisher D_{il} runs $\widetilde{D}$, and handles $\widetilde{D}$'s queries as follows:

- $\widetilde{D}$'s P- and H-queries are simply relayed;
- for each E-query $\mathrm{E}^{\delta}(K, z)$ from $\widetilde{D}$, D_{il} "illegally" accesses the randomness source $\mathbf{R}$ to get $k \leftarrow \mathbf{R}.\mathrm{H}(K)$ and answers with $k \oplus \mathrm{E}^{\delta}(K, k \oplus z)$.

By construction, it always holds $HQueries \cong \mathbf{R}$ for the set $HQueries$ of S (even if S aborts). Thus the execution of $\widetilde{G}_2(\mathbf{E}, \mathbf{R})$ and $\widetilde{G}_3(\mathbf{E}, \mathbf{R})$ are essentially the same, and $Pr[\widetilde{G}_2 = 1] = Pr[\widetilde{G}_3 = 1]$. On the other hand, note that the total number of queries issued by D_{il} to S and $\mathbf{E}$ is the same as $\widetilde{D}$.

$\widetilde{G}_4$: this imagined game takes $\mathbf{R}$ as the randomness source, and captures the interaction between the distinguisher D_{il}, the cipher EMR_3^*, and the random primitives $\mathbf{R}$. Since D_{il} issues the same number of queries as $\widetilde{D}$, by Theorem 2 we get $|Pr[\widetilde{G}_4 = 1] = Pr[\widetilde{G}_3 = 1]| \leq \frac{2514 q_h^6 (q_e+q_p)^2 \cdot q_p^4}{N} + \frac{2359 q_e^2 (q_e+q_p)^2 \cdot q_p^4}{N} + \frac{338 q_e \cdot q_h (q_e+q_p)^2 \cdot q_p^4}{N} + \frac{q_h^2 + q_h^4 + 8q_e^2 + q_e \cdot q_h}{N}$.

$\widetilde{G}_5$: this game takes $\mathbf{R}$ as the randomness source, and captures the interaction between the distinguisher $\widetilde{D}$, the cipher EMR_3, and the random primitives $\mathbf{R}$. Clearly $Pr[\widetilde{G}_5 = 1] = Pr[\widetilde{G}_4 = 1]$. Finally, it's easy to see $\mathrm{Adv}^{indif}_{\mathsf{EMR}_3, \mathbf{E}, \widetilde{S}} = |Pr[\widetilde{G}_5 = 1] - Pr[\widetilde{G}_1 = 1]|$. Thus the claim.

Discussion. Given $\widetilde{D}$ on EMR_3, consider a distinguisher D, which runs $\widetilde{D}$ and handles $\widetilde{D}$'s queries as follows:

- $\widetilde{D}$'s P- and H-queries are simply relayed;
- for each E-query $\mathrm{E}^{\delta}(K, z)$ from $\widetilde{D}$, D queries the right oracle $\mathrm{H} \rightarrow k$ and answers with $k \oplus \mathrm{E}^{\delta}(K, k \oplus z)$.

Clearly D is a distinguisher on EMR_3^*, and is sufficient to prove EMR_3 indifferentiable. However, D issues at most $q_e + q_h$ queries to H in total, and this would bring in an uncomfortable security loss. This explains why we take a quite complicated sequence of games as above, which involves the illegal distinguisher D_{il}.

13 Eliminating the Random Oracle: to EMDP_3

The second result of this work is formally presented as follows.

Theorem 3. *Assuming that* $\mathbf{P}$ *is a tuple of four independent random permutations. Then for the 3-round Even-Mansour*

$$\mathit{EMDP}_3(K, m) = k \oplus \mathbf{P}_3(k \oplus \mathbf{P}_2(k \oplus \mathbf{P}_1(k \oplus m)))$$

with $k = \mathbf{P}_0(K) \oplus K$*, there exists a simulator* $\mathcal{S}$ *such that* $Adv^{indif}_{\mathit{EMDP}_3, \mathbf{E}, \mathcal{S}} \leq O(\frac{q^{12}}{N})$ *for any distinguisher* D *that makes at most* q *queries (here* $\mathbf{E}$ *stands for ideal* (n, n)*-ciphers). Moreover,* $\mathcal{S}$ *makes at most* $O(q^4)$ *queries to* $\mathbf{E}$*, and runs in time* $O(q^7)$*.*

We then brief how to modify EMR_3's simulator S for Theorem 3. However, to save pages, we did not try to work out all the concrete constant factors of Theorem 3.

Modified Simulator $\mathcal{S}^{\mathbf{E},\mathbf{P}}$ We let the simulator $\mathcal{S}$'s randomness source $\mathbf{P}$ supply two additional interfaces P0 and $\mathrm{P0}^{-1}$. The interface provided by $\mathcal{S}$ is exactly the same as $\mathbf{P}$. The overall strategy of $\mathcal{S}$ is very close to that of S – except for replacing the procedure H by P0 and $\mathrm{P0}^{-1}$. The modifications around P0 is described as the following pseudocode.

Simulated System G_1'
Variables
// The same as G_1 in subsection 5.2, thus omitted.

```
public procedure P0(x_0)
  if x_0 ∈ P_0 then return P_0(x_0)
  y_0 ← P.P0(x_0)
  K ← x_0, k ← x_0 ⊕ y_0
  if k ∈ Z then abort
  if ∃k', k'', k''' ∈ Z : k ⊕ k' ⊕ k'' ⊕ k''' = 0 then abort
  if ∃i, y_i ∈ P_i^{-1}, x_{i+1} ∈ P_{i+1} : y_i ⊕ x_{i+1} ∈ (k ⊕ 4Z) ∪ {k}
    or ∃i, x_i, x'_i ∈ P_i : x_i ⊕ x'_i ∈ k ⊕ 5Z
    or ∃i, y_i, y'_i ∈ P_i^{-1} : y_i ⊕ y'_i ∈ k ⊕ 5Z then abort
  Queries ← Queries ∪ {(0, x_0, y_0)}
  HQueries ← HQueries ∪ {(K, k, qnum)}
  qnum ← qnum + 1
  foreach (1, x_1, y_1), (3, x_3, y_3) ∈ Queries do
    if Check(K, x_1, y_3) = true then
      Take (K, x_1, y_3, edir, enum) from EQueries
      Adapt(2, y_1 ⊕ k, x_3 ⊕ k, edir, enum)
      // "Dummy" edir and enum in G'_1.
      Assert(∀k' ∈ Z\{k} : x_2 ⊕ k' ∉ P_1^{-1} and y_2 ⊕ k' ∉ P_3)
      UpdateCompleted(1, K, x_1)
  // Update AD2Edges: same as G_1, omitted
  return P_0(x_0)
```

```
public procedure P0^{-1}(y_0)
  if y_0 ∈ P_0^{-1} then return P_0^{-1}(y_0)
  x_0 ← P.P0^{-1}(y_0)
  K ← x_0, k ← x_0 ⊕ y_0
  if k ∈ Z then abort
  if ∃k', k'', k''' ∈ Z : k ⊕ k' ⊕ k'' ⊕ k''' = 0 then abort
  if ∃i, y_i ∈ P_i^{-1}, x_{i+1} ∈ P_{i+1} : y_i ⊕ x_{i+1} ∈ (k ⊕ 4Z) ∪ {k}
    or ∃i, x_i, x'_i ∈ P_i : x_i ⊕ x'_i ∈ k ⊕ 5Z
    or ∃i, y_i, y'_i ∈ P_i^{-1} : y_i ⊕ y'_i ∈ k ⊕ 5Z then abort
  Queries ← Queries ∪ {(0, x_0, y_0)}
  HQueries ← HQueries ∪ {(K, k, qnum)}
  qnum ← qnum + 1
  foreach (1, x_1, y_1), (3, x_3, y_3) ∈ Queries do
    if Check(K, x_1, y_3) = true then
      Take (K, x_1, y_3, edir, enum) from EQueries
      Adapt(2, y_1 ⊕ k, x_3 ⊕ k, edir, enum)
      // "Dummy" edir and enum in G'_1.
      Assert(∀k' ∈ Z\{k} : x_2 ⊕ k' ∉ P_1^{-1} and y_2 ⊕ k' ∉ P_3)
      UpdateCompleted(1, K, x_1)
  // Update AD2Edges: same as G_1, omitted
  return P_0^{-1}(y_0)
```

Discussion. Since EMDP_3 has the whitening keys, the mechanism for H-TPs can be replaced by abortion checks, i.e. if a newly derived round-key k links pre-existing E-queries and 1-/3-queries, then G'_2 aborts. E.g. $\exists (K, y_0, x_4)$ and $(1, x_1, y_1)$ with $k = y_0 \oplus x_1$. However, to keep the bounds at the same order as Theorem 1, we do not incorporate this change.

14 Implication on Multiple Known-Key Indifferentiability of 3-round Even-Mansour

Formally, the (n, n)-blockcipher SEM_3 is

$$\mathsf{SEM}_3(k, m) = k \oplus \mathbf{P}_3(k \oplus \mathbf{P}_2(k \oplus \mathbf{P}_1(k \oplus m))),$$

as depicted in Fig. 13. There's no KD.

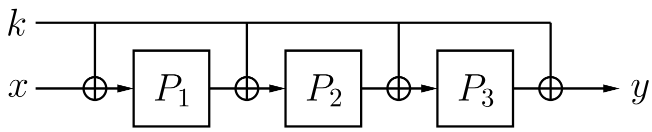


Fig. 13. The 3-round single-key Even-Mansour

The main result in this section is formally stated as follows:

Theorem 4. *Assuming that* $\mathbf{P}$ *is a tuple of three independent random permutations, and consider the* (n, n)*-blockcipher* SEM_3 *built from* $\mathbf{P}$*. Then for any* ζ*, under* ζ *random known-keys, there exists a simulator* S_{KK} *such that*

$$Adv^{indif}_{\mathsf{SEM}_3, \mathbf{E}, S_{KK}} \le \frac{2514\zeta^6 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{1787 q_e^2 (q_e + q_p)^2 \cdot q_p^4}{N} + \frac{\zeta^4 + 7q_e^2}{N}$$

for any distinguisher D *that makes at most* q_e *and* q_p *queries to the (fixed-key) encryption/decryption oracle and the random permutations respectively. Moreover,* S_{KK} *makes at most* $26\zeta \cdot (q_e + q_p) \cdot q_p^2$ *queries to the ideal* (n, n)*-blockcipher* $\mathbf{E}$ *and runs in time* $O((q_e + q_p)^2 \cdot q_p^4 + \zeta (q_e + q_p)^2 \cdot q_p^4)$.

The simulator S_{KK} is built from S of section 5, in an almost-black-box manner. At the beginning of the interaction, S_{KK} checks the set $\mathcal{K}$ of ζ known-keys $k_1, \ldots, k_\zeta$. If there exist four distinct keys $k, k', k'', k''' \in \mathcal{K}$ such that $k \oplus k' \oplus k'' \oplus k''' = 0$, S_{KK} aborts. Otherwise, it runs an instance of the simulator S for EMR_3 – but it enforce the set $HQueries$ of S to contain ζ tuples $(k_1, k_1), \ldots, (k_\zeta, k_\zeta)$. It then answers D's queries with S's interfaces, and aborts whenever S aborts. It's not hard to see that this experiment is equivalent to D first issuing ζ H-queries and then issuing the others. Thus the claim.

However, to calculate the indifferentiability bound, we should replace q_h in the bound of Theorem 1 by ζ, and subtract the following terms from it:

(i) The term $\frac{\zeta^2}{N}$. The "original term" $\frac{q_h^2}{N}$ is introduced due to the bad event of two distinct main-keys being mapped to the same round-key. However, the ζ known-keys are ensured to be distinct.

(ii) The terms $\frac{(q_e+\zeta)\cdot q_e+234q_e(q_e+q_p)^2\cdot q_p^4+338q_e(q_e+\zeta)(q_e+q_p)^2\cdot q_p^4}{N}$. The "original version" of them are introduced by the possibility of the random oracle being a bad one for G_3, cf. Lemma 22. In the context of this section, these "bad events" have no chance to occur.

Having the above subtracted, we got the bound $\frac{2514\zeta^6(q_e+q_p)^2\cdot q_p^4}{N}+\frac{1787q_e^2(q_e+q_p)^2\cdot q_p^4}{N}+\frac{\zeta^4+7q_e^2}{N}$.

Discussion. For the term $\frac{\zeta^4}{N}$, we have the following discussion. Among the ζ known-keys, if there exist four distinct keys k_1, k_2, k_3, k_4 such that $k_1 \oplus k_2 \oplus k_3 \oplus k_4 = 0$, then *our simulator is not applicable.* We stress that this does not necessarily means SEM_3 is not indifferentiable in this case; it only means *we should turn to some other simulator to completely solve this* – indeed, we conjecture that SEM_3 is indifferentiable under *any* set of ζ known-keys, i.e. ζ-KK-indifferentiable [CS16]. On the other hand, if the known-keys are ensured not contain such four keys, then (informally) SEM_3 is $(q, O(\zeta \cdot q^3), O(\zeta \cdot q^6), O\left(\frac{\zeta^6 \cdot q^6+q^8}{N}\right))$-indifferentiable. In particular, when $\zeta \leq 3$, e.g. building compression functions from three permutations [MP12], our analysis ensures the known-key security of SEM_3.

Acknowledgements

We thank Bart Mennink for insightful feedback. We also thank Yu Yu for helpful suggestions.

This work is partially supported by National Key Basic Research Project of China (2011CB302400), National Science Foundation of China (61379139) and the "Strategic Priority Research Program" of the Chinese Academy of Sciences, Grant No. XDA06010701.

A On Eliminating Whitening-Keys

In this section, we exhibit an artificial simulator S^* for EMR_3^*, which is effective but cannot be used by the argument in Section 12. Basically, S^* is built from the successful simulator S, with some additional silly actions which do not harm the effectiveness but hinder the argument in Section 12. To wit, S^* runs S: upon each query $\mathrm{H}(K)$, S^* first internally samples a random pair (K', x') and makes a "dummy" query $\mathrm{E}(K', x')$ to $\mathbf{E}$, and then answers $\mathrm{H}(K)$ with $S.\mathrm{H}(K)$. It's clear that: (i) this simulator works as well as S, except for making q_h additional dummy queries to $\mathbf{E}$; (ii) w.h.p. $K' \notin S.HTable$ before S^* queries $\mathrm{E}(K', x')$, and thus the approach in Section 12 cannot be used to build $\widetilde{S}$ from S^*.

If we slightly tweak the strategy of $\widetilde{S}$ by letting it query $S^*.\mathrm{H}(K')$ for k' and answer with $k' \oplus \mathrm{E}(K', k' \oplus x')$, then S^* would pushes another dummy query $\mathrm{E}(K'', x'')$. In such a way, the interaction would *run forever.* In all, while we strongly believe EMR_t and EMR_t^* have the same indifferentiability security (regardless of t's value), we did not find a general proof for this transformation.

B Keeping P_2 Random is an Impossible Mission

The aforementioned distinguisher D works as follows:

(1) Chooses $x_1 \in \{0,1\}^n$, 6 distinct main-keys $K_1, K_2, \ldots, K_6 \in \{0,1\}^\kappa$, and queries $\mathrm{H}(K_i) \to k_i$ for $i = 1, \ldots, 6$, $\mathrm{P1}(x_1) \to y_1$;

(2) Makes 6 queries to E and E^{-1}: $\mathrm{E}(K_1, x_1) \to y_1$, $\mathrm{E}^{-1}(K_2, y_1) \to x_1'$, $\mathrm{E}(K_3, x_1') \to y_1'$, $\mathrm{E}^{-1}(K_4, y_1') \to x_1''$, $\mathrm{E}(K_5, x_1'') \to y_1''$, and $\mathrm{E}^{-1}(K_6, y_1'') \to x_1'''$;

(3) Queries $\mathrm{P1}(x_1''') \to y_1'''$;

(4) Completes the six chains corresponding to (K_1, x_1, y_1), (K_2, x_1', y_1), (K_3, x_1', y_1'), (K_4, x_1'', y_1'), (K_5, x_1'', y_1''), and (K_6, x_1''', y_1'').

Now the simulator has to adapt six chains. However, after it completes step (3), there are only *five* 1- and 3-queries that can be defined as adapted ones, i.e. $(3, x_3, y_3)$, $(1, x_1', y_1')$, $(3, x_3', y_3')$, $(1, x_1'', y_1'')$, and $(3, x_3'', y_3'')$. The simulator thus cannot settle all the six chains, and has to use P_2 for adaptation.

Here we only give an instructive example. The distinguisher could indeed choose q main-keys and make q queries to E and E^{-1}. If the simulator wants to "go ahead" to keep P_2 "random", then it probably has to make $O(q^q)$ queries to $\mathrm{E}/\mathrm{E}^{-1}$. Keeping P_2 random is thus not possible.

References

ABD+13a. Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the Indifferentiability of Key-Alternating Ciphers. Cryptology ePrint Archive, Report 2013/061, 2013. Extended abstract appeared at CRYPTO 2013.

ABD+13b. Elena Andreeva, Andrey Bogdanov, Yevgeniy Dodis, Bart Mennink, and John P. Steinberger. On the Indifferentiability of Key-Alternating Ciphers. Cryptology ePrint Archive, Report 2013/061, 2013. Version: 20130206:161230.

ABK98. Ross Anderson, Eli Biham, and Lars Knudsen. Serpent: A Proposal for the Advanced Encryption Standard. *NIST AES Proposal*, 174, 1998.

ABM13. Elena Andreeva, Andrey Bogdanov, and Bart Mennink. Towards Understanding the Known-Key Security of Block Ciphers. In Shiho Moriai, editor, *Fast Software Encryption*, Lecture Notes in Computer Science, pages 348–366. Springer Berlin Heidelberg, 2013.

Bih94. Eli Biham. New Types of Cryptanalytic Attacks Using Related Keys. *Journal of Cryptology*, 7(4):229–246, 1994.

BKL+12. Andrey Bogdanov, LarsR. Knudsen, Gregor Leander, François-Xavier Standaert, John Steinberger, and Elmar Tischhauser. Key-Alternating Ciphers in a Provable Setting: Encryption Using a Small Number of Public Permutations. In David Pointcheval and Thomas Johansson, editors, *Advances in Cryptology – EUROCRYPT 2012*, volume 7237 of *Lecture Notes in Computer Science*, pages 45–62. Springer Berlin Heidelberg, 2012.

Bla06. John Black. The Ideal-Cipher Model, Revisited: An Uninstantiable Blockcipher-Based Hash Function. In Matthew Robshaw, editor, *Fast Software Encryption*, volume 4047 of *Lecture Notes in Computer Science*, pages 328–340. Springer Berlin Heidelberg, 2006.

CDMP05. Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård Revisited: How to Construct a Hash Function. In Victor Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, pages 430–448. Springer Berlin Heidelberg, 2005.

CGH04. Ran Canetti, Oded Goldreich, and Shai Halevi. The Random Oracle Methodology, Revisited. *J. ACM*, 51(4):557–594, July 2004.

CHK+16. Jean-Sébastien Coron, Thomas Holenstein, Robin Künzler, Jacques Patarin, Yannick Seurin, and Stefano Tessaro. How to Build an Ideal Cipher: The Indifferentiability of the Feistel Construction. *Journal of Cryptology*, 29(1):61–114, 2016.

CLL+14. Shan Chen, Rodolphe Lampe, Jooyoung Lee, Yannick Seurin, and John Steinberger. Minimizing the Two-Round Even-Mansour Cipher. In JuanA. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, volume 8616 of *Lecture Notes in Computer Science*, pages 39–56. Springer Berlin Heidelberg, 2014. full version: https://eprint.iacr.org/2014/443.pdf.

CLS15. Benoît Cogliati, Rodolphe Lampe, and Yannick Seurin. Tweaking Even-Mansour Ciphers. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 189–208. Springer Berlin Heidelberg, 2015. full version: http://eprint.iacr.org/2015/539.pdf.

CPS08. Jean-Sébastien Coron, Jacques Patarin, and Yannick Seurin. The Random Oracle Model and the Ideal Cipher Model Are Equivalent. In David Wagner, editor, *Advances in Cryptology – CRYPTO 2008*, volume 5157 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg, 2008.

CS14. Shan Chen and John Steinberger. Tight Security Bounds for Key-Alternating Ciphers. In PhongQ. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 327–350. Springer Berlin Heidelberg, 2014. full version: https://eprint.iacr.org/2013/222.pdf.

CS15a. Benoît Cogliati and Yannick Seurin. Beyond-Birthday-Bound Security for Tweakable Even-Mansour Ciphers with Linear Tweak and Key Mixing. In Tetsu Iwata and JungHee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, pages 134–158. Springer Berlin Heidelberg, 2015.

CS15b. Benoît Cogliati and Yannick Seurin. On the Provable Security of the Iterated Even-Mansour Cipher Against Related-Key and Chosen-Key Attacks. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015*, volume 9056 of *Lecture Notes in Computer Science*, pages 584–613. Springer Berlin Heidelberg, 2015.

CS16. Benoît Cogliati and Yannick Seurin. Strengthening the Known-Key Security Notion for Block Ciphers. In *Fast Software Encryption 2016*, Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2016.

Dae93. Joan Daemen. Limitations of the Even-Mansour construction. In Hideki Imai, RonaldL. Rivest, and Tsutomu Matsumoto, editors, *Advances in Cryptology – ASIACRYPT '91*, volume 739 of *Lecture Notes in Computer Science*, pages 495–498. Springer Berlin Heidelberg, 1993.

DDKS14. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Cryptanalysis of Iterated Even-Mansour Schemes with Two Keys. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, volume 8873 of *Lecture Notes in Computer Science*, pages 439–457. Springer Berlin Heidelberg, 2014.

DDKS16. Itai Dinur, Orr Dunkelman, Nathan Keller, and Adi Shamir. Key Recovery Attacks on Iterated Even-Mansour Encryption Schemes. *Journal of Cryptology*, 29(4):697–728, 2016.

DGHM13. Grégory Demay, Peter Gaži, Martin Hirt, and Ueli Maurer. Resource-Restricted Indifferentiability. In Thomas Johansson and PhongQ. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 664–683. Springer Berlin Heidelberg, 2013.

DKS15. Orr Dunkelman, Nathan Keller, and Adi Shamir. Slidex Attacks on the Even-Mansour Encryption Scheme. *Journal of Cryptology*, 28(1):1–28, 2015.

DRST12. Yevgeniy Dodis, Thomas Ristenpart, John Steinberger, and Stefano Tessaro. To Hash or Not to Hash Again? (In)Differentiability Results for H^2 and HMAC. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 348–366. Springer Berlin Heidelberg, 2012.

DS16. Yuanxi Dai and John Steinberger. Indifferentiability of 8-Round Feistel Networks. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 95–120. Springer Berlin Heidelberg, 2016.

DSKT16. Dana Dachman-Soled, Jonathan Katz, and Aishwarya Thiruvengadam. 10-Round Feistel is Indifferentiable from an Ideal Cipher. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 649–678. Springer Berlin Heidelberg, 2016.

DSSL16. Yevgeniy Dodis, Martijn Stam, John Steinberger, and Tianren Liu. Indifferentiability of Confusion-Diffusion Networks. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 679–704. Springer Berlin Heidelberg, 2016.

EM97. Shimon Even and Yishay Mansour. A Construction of a Cipher From a Single Pseudorandom Permutation. *Journal of Cryptology*, 10(3):151–161, 1997.

FP15. Pooya Farshim and Gordon Procter. The Related-Key Security of Iterated Even-Mansour Ciphers. In Gregor Leander, editor, *Fast Software Encryption*, volume 9054 of *Lecture Notes in Computer Science*, pages 342–363. Springer Berlin Heidelberg, 2015. full version: http://eprint.iacr.org/2014/953.pdf.

GL15a. Chun Guo and Dongdai Lin. A Synthetic Indifferentiability Analysis of Interleaved Double-Key Even-Mansour Ciphers. In Tetsu Iwata and JungHee Cheon, editors, *Advances in Cryptology – ASIACRYPT 2015*, volume 9453 of *Lecture Notes in Computer Science*, pages 389–410. Springer Berlin Heidelberg, 2015.

GL15b. Chun Guo and Dongdai Lin. On the Indifferentiability of Key-Alternating Feistel Ciphers with No Key Derivation. In Yevgeniy Dodis and JesperBuus Nielsen, editors, *Theory of Cryptography*, volume 9014 of *Lecture Notes in Computer Science*, pages 110–133. Springer Berlin Heidelberg, 2015. full version: http://eprint.iacr.org/.

GL15c. Chun Guo and Dongdai Lin. Separating Invertible Key Derivations from Non-invertible Ones: Sequential Indifferentiability of 3-round Even-Mansour. *Designs, Codes and Cryptography*, pages 1–21, 2015.

HKT11. Thomas Holenstein, Robin Künzler, and Stefano Tessaro. The Equivalence of the Random Oracle Model and the Ideal Cipher Model, Revisited. In *Proceedings of the Forty-third Annual ACM Symposium on Theory of Computing – STOC '11*, pages 89–98, New York, NY, USA, 2011. ACM.

HT16. Viet Tung Hoang and Stefano Tessaro. Key-Alternating Ciphers and Key-Length Extension: Exact Bounds and Multi-user Security. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 3–32. Springer Berlin Heidelberg, 2016.

LPS12. Rodolphe Lampe, Jacques Patarin, and Yannick Seurin. An Asymptotically Tight Security Analysis of the Iterated Even-Mansour Cipher. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *Lecture Notes in Computer Science*, pages 278–295. Springer Berlin Heidelberg, 2012.

LRW11. Moses Liskov, RonaldL. Rivest, and David Wagner. Tweakable Block Ciphers. *Journal of Cryptology*, 24(3):588–613, 2011.

LS13. Rodolphe Lampe and Yannick Seurin. How to Construct an Ideal Cipher from a Small Set of Public Permutations. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology – ASIACRYPT 2013*, volume 8269 of *Lecture Notes in Computer Science*, pages 444–463. Springer Berlin Heidelberg, 2013.

Men15. Bart Mennink. XPX: Generalized Tweakable Even-Mansour with Improved Security Guarantees. Cryptology ePrint Archive, Report 2015/476, 2015. Extended abstract to appear at CRYPTO 2016.

ML15. Nicky Mouha and Atul Luykx. Multi-key Security: The Even-Mansour Construction Revisited. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology – CRYPTO 2015*, volume 9215 of *Lecture Notes in Computer Science*, pages 209–223. Springer Berlin Heidelberg, 2015.

MP12. Bart Mennink and Bart Preneel. Hash Functions Based on Three Permutations: A Generic Security Analysis. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, volume 7417 of *Lecture Notes in Computer Science*, pages 330–347. Springer Berlin Heidelberg, 2012.

MPS12. Avradip Mandal, Jacques Patarin, and Yannick Seurin. On the Public Indifferentiability and Correlation Intractability of the 6-Round Feistel Construction. In Ronald Cramer, editor, *Theory of Cryptography*, volume 7194 of *Lecture Notes in Computer Science*, pages 285–302. Springer Berlin Heidelberg, 2012.

MRH04. Ueli Maurer, Renato Renner, and Clemens Holenstein. Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology. In Moni Naor, editor, *Theory of Cryptography*, volume 2951 of *Lecture Notes in Computer Science*, pages 21–39. Springer Berlin Heidelberg, 2004.

NWW13. Ivica Nikolić, Lei Wang, and Shuang Wu. Cryptanalysis of Round-Reduced LED. In Shiho Moriai, editor, *Fast Software Encryption*, Lecture Notes in Computer Science, pages 112–129. Springer Berlin Heidelberg, 2013.

Pat09. Jacques Patarin. *Selected Areas in Cryptography: 15th International Workshop, SAC 2008, Sackville, New Brunswick, Canada, August 14-15, Revised Selected Papers*, chapter The "Coefficients H" Technique, pages 328–345. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009.

RSS11. Thomas Ristenpart, Hovav Shacham, and Thomas Shrimpton. Careful with Composition: Limitations of the Indifferentiability Framework. In KennethG. Paterson, editor, *Advances in Cryptology – EUROCRYPT 2011*, volume 6632 of *Lecture Notes in Computer Science*, pages 487–506. Springer Berlin Heidelberg, 2011.

Ste12. John Steinberger. Improved Security Bounds for Key-Alternating Ciphers via Hellinger Distance. Cryptology ePrint Archive, Report 2012/481, 2012. http://eprint.iacr.org/2012/481.pdf.