BiTR: Built-in Tamper Resilience

Seung Geol Choi* Aggelos Kiayias† Tal Malkin‡

January 12, 2012

Abstract

The assumption of the availability of tamper-proof hardware tokens has been used extensively in the design of cryptographic primitives. For example, Katz (Eurocrypt 2007) suggests them as an alternative to other setup assumptions, towards achieving general UC-secure multi-party computation. On the other hand, a lot of recent research has focused on protecting security of various cryptographic primitives against physical attacks such as leakage and tampering.

In this paper we put forward the notion of Built-in Tamper Resilience (BiTR) for cryptographic protocols, capturing the idea that the protocol that is encapsulated in a hardware token is designed in such a way so that tampering gives no advantage to an adversary. Our definition is within the UC model, and can be viewed as unifying and extending several prior related works. We provide a composition theorem for BiTR security of protocols, impossibility results, as well as several BiTR constructions for specific cryptographic protocols or tampering function classes. In particular we achieve general UC-secure computation based on a hardware token that may be susceptible to affine tampering attacks. We also present BiTR proofs for identification and signature schemes in the same tampering model. We next observe that non-malleable codes are sufficient (but not necessary) as state encodings to imply the BiTR property and we show new positive results for deterministic non-malleable encodings (as opposed to probabilistic that were previously known) for various classes of tampering functions.

1 Introduction

Security Against Physical Attacks. Traditionally, cryptographic schemes have been analyzed assuming that an adversary has only *black-box* access to the underlying functionality, and no way to manipulate the internal state. For example, traditional security definitions for encryption schemes address an adversary who is given the public key — but not the private key — and tries to guess something about the plaintext of a challenge ciphertext, by applying some black-box attack (e.g., CPA or CCA). In practical situations, however, an adversary can often do more. For example, when using small portable devices such as smart-cards or mobile-phones, an adversary can take hold of the device and apply a battery of attacks. One class of attacks are those that try to recover information via side channels such as power consumption [KJJ99], electromagnetic radiation [QS01], and timing [BB05]. To address these attacks, starting with the work of [ISW03, MR04] there has

*University of Maryland sgchoi@cs.umd.edu

†University of Connecticut aggelos@cse.uconn.edu. Supported in part by NSF grants 0447808, 0831304, and 0831306.

‡Columbia University tal@cs.columbia.edu. Supported in part by NSF grants 0831094 and 0347839.

been a surge of recent research activity on leakage-resilient cryptographic schemes. For example, refer to [SMY09, AGV09, DKL09, Pie09, NS09, ADW09, KV09, FKPR10, DGK+10, FRR+10, BG10, DP10, JV10, GR10, LW10, BKKV10, P11, MTVY11, HL11].

The present work addresses *tampering attacks*, where an adversary can modify the secret data by applying various physical attacks (c.f., [AK96, BDL01, BS97, SA02, Sko05, BECN+04]). Currently, there are only a few results in this area [GLM+04, IPSW06, DPW10].

Hardware Tokens. As discussed above, cryptographic primitives have traditionally been assumed to be tamper (and leakage) proof. In the context of larger cryptographic protocols, there have been many works that (implicitly or explicitly) used secure hardware as a tool to achieve security goals that could not be achieved otherwise. The work most relevant to ours is that of Katz [Kat07], who suggests to use *tamper-proof hardware tokens* to achieve UC-secure [Can01] commitments. This allows achieving general feasibility results for UC-secure well-formed multi-party computation, where the parties, without any other setup assumptions, send each other tamper-proof hardware tokens implementing specific two-party protocols. There were several follow-up works such as [MS08, CGS08, DNW08, GIS+10, Kol10, GIMS10, DKM11], all of which assume a token that is tamper proof.

Given the wide applicability of tamper-proof tokens on one hand, and the reality of tampering attacks on the other, we ask the following natural question:

> *Can we relax the tamper-proof assumption, and get security using* tamperable *hardware tokens?*

Clearly, for the most general interpretation of this question, the answer is typically negative. For example, if the result of [Kat07] was achievable with arbitrarily-tamperable hardware token, that would give general UC-secure protocols in the "plain" model, which is known to be impossible [CF01]. In this work we address the above question in settings where the class of possible tampering functions and the class of protocols we wish to put in a token and protect are restricted.

1.1 Our Contributions

BiTR Definition. We provide a definition of Built-in Tamper Resilience (BiTR) for two party cryptographic protocols, capturing the idea that the protocol can be encapsulated in a hardware token, whose state may be tamperable. Our definition is very general, compatible with the UC setting [Can01], and implies that any BiTR protocol can be used as a hardware token within larger UC-protocols. Our definition may be viewed as unifying and generalizing previous definitions [GLM+04, IPSW06, DPW10] and bringing them to the UC setting, as well as bringing aspects of notions such as related key security for PRFs [BK03] to the setting of any cryptographic primitive (see Section 1.2).

BiTR is a property of a cryptographic protocol M, which roughly says the following. Any adversary that is able to apply tampering functions from the class $\mathcal{T}$ on a token running M, can be simulated by an adversary that has no tampering capability, independently of the environment in which the tokens may be deployed.

The strongest result one would ideally want is a general compiler that takes an arbitrary protocol and transforms it to an equivalent protocol that is BiTR against arbitrary tampering functions, without having to encode the state into a larger one, and without requiring any additional ran-

domness.[1] Since such a strong result is clearly impossible, we provide several specific results that trade off these parameters (see below), as well as the following composition theorem.

BiTR Composition. As BiTR is a protocol centric property, the natural question that arises is whether it is preserved under composition. A useful result for a general theory of BiTR cryptography would be a general composition theorem which allows combining a BiTR protocol calling a subroutine and a BiTR implementation of that subroutine into one overall BiTR protocol. To this end, we characterize BiTR composition of protocols by introducing the notion of modular-BiTR which captures the property of being BiTR in the context of a larger protocol. We then prove that the property of modular-BiTR is *necessary and sufficient* for construction of composite BiTR protocols. At the same time we also derive a negative result, namely that modular-BiTR protocols that preserve the BiTR property in any possible context (something we term universal-BiTR) are unattainable assuming the existence of one-way functions, at least for non-trivial protocols. These results thus settle the question of BiTR composability.

BiTR Constructions without State Encoding. We describe results for BiTR primitives that require no state encodings. It may come as a surprise that it is possible to prove a cryptographic protocol BiTR without any encoding and thus without any validation of the secret protocol state whatsoever. This stems from the power of our definitional framework for BiTR and the fact that it is can be achieved for specially selected and designed protocols and classes of tampering functions. We define the class $\mathcal{T}_{\mathsf{aff}} = \{f_{a,b} \mid a \in \mathbb{Z}_q^*, b \in \mathbb{Z}_q, f_{a,b}(v) := av + b \bmod q\}$. That is, the adversary may apply a modular affine function of his choice to tamper the state. Affine tampering is an interesting class to consider as it has as special cases multiplication (e.g., shifting — which may be the result of tampering shift-register based memory storage), or addition (which may be result of bit flipping tampering).

We prove three protocols BiTR with respect to this class, where the tamper resilience is really "built-in" in the sense that no modification of the protocol or encoding of the state are necessary. The first one is Schnorr's identification (two-round) protocol [Sch91]. The second is Okamoto's signature scheme [Oka06]. Both protocols are interesting on their own (e.g., previous work [GLM+04] focused mostly on signature schemes), but the latter is also useful for the third protocol we prove affine-BiTR, described next.

UC-Secure Computation from tamperable tokens. Katz's approach [Kat07] for building UC-secure computation using hardware tokens allows a natural generalization that involves a commitment scheme with a special property, we call a *dual-mode parameter generation (DPG)* — depending on the mode of the parameter, the commitment scheme is either statistically hiding or a trapdoor commitment. We then observe that any DPG-commitment is sufficient for providing UC-secure multi-party computation assuming tamper proof tokens. Following this track, we present a new DPG-commitment scheme that is BiTR against affine tampering functions, that relies on discrete-log based primitives including the digital signature scheme of Okamoto [Oka06]. Thus, we obtain UC-secure general computation using hardware tokens tamperable with affine functions. We also examine a different class of tokens that implement a single OT [GKR08] and were utilized in [GIS+10] for UC secure computation. We present tampering functions against which these tokens are BiTR.

[1] If an encoding ψ of the state is required, it is desirable that it is deterministic (randomness may not be available in some systems or expensive to generate), and that it has as high rate as possible. Ideally, an existing scheme can be proven BiTR *as-is*, without any state encoding at all.

BiTR Constructions with State Encoding. We next discuss how one can take advantage of state consistency checks to design BiTR protocols. We observe first that non-malleable codes, introduced by Dziembowski, Pietrzak and Wichs [DPW10] can be used as an encoding for proving the BiTR property of protocols. This gives rise to the problem of constructing such codes. Existing constructions [DPW10] utilize randomness in calculating the encoding; we provide new constructions for such encodings focusing on purely *deterministic constructions*. In fact, when the protocol uses no randomness (e.g., a deterministic signing algorithm) or a finite amount of randomness (e.g., a prover in the resettable zero-knowledge [CGGM00] setting), by using deterministic encodings the token may dispense with the need of random number generation.

Our design approach takes advantage of a generalization of non-malleable encodings (called δ-non-malleable), and we show how they can be constructible for any given set of tampering functions (as long as they exist). Although inefficient for general tampering functions, the construction becomes useful if each function in the class $\mathcal{T}$ works independently on small blocks (of logarithmic size). In this case, we show that a non-malleable code for the overall state can be constructed efficiently by first applying Reed-Solomon code to the overall state and then applying δ-non-malleable codes for small blocks to the resulting codeword. We stress that this construction is intended as a feasibility result.

1.2 Related Work

We briefly describe the most relevant previous works addressing protection against tampering. We note that none of these works had addressed tampering in the context of UC-secure protocols.

Gennaro et al. [GLM+04] considered a device with two separate components: one is tamper-proof yet readable (circuitry), and the other is tamperable yet read-proof (memory). They defined algorithmic tamper-proof (ATP) security and explored its possibility for signature and decryption devices. Their definition of ATP security was given only for the specific tasks of signature and encryption. In contrast, our definition is simulation based, independent of the correctness or security objectives of the protocol, and we consider general two-party protocols (and the implications in the UC framework [Can01, Kat07]).

Ishai et al. [IPSW06] considered an adversary who can tamper with the wires of a circuit. They showed a general compiler that outputs a self-destructing circuit that withstands such a tampering adversary. Considering that memory corresponds to a subset of the wires associated with the state in their model, the model seems stronger than ours (as we consider only the state, not the computation circuit). However, the tampering attack they considered is very limited: it modifies a bounded subset of the wires between each invocation, which corresponds to tampering memory only partially.

Dziembowski et al. [DPW10] introduced the notion of non-malleable codes and tamper simulatability to address similar concerns as the present work. A distinguishing feature of BiTR security from their approach is that BiTR is protocol-centric. As such, it allows arguing about tamper resilience by taking advantage of specific protocol design features that enable BiTR even without any encodings. Moreover, the positive results of [DPW10] require the introduction of additional circuitry or a randomness device; this may be infeasible, uneconomical or even unsafe in practice — it could be introducing new pathways for attacks. In contrast, our positive results do not require state encodings or when they do, they do not rely on randomness.

Bellare and Kohno defined security against related key attacks (RKA) for block ciphers [BK03], and there has been follow-up work [BC10a, AHI11] (see also the references therein). Roughly

speaking, RKA-security as it applies to PRFs and encryption is a strengthening of the security definition of the underlying primitive (be it indistinguishability from random functions or semantic security). RKA-security was only shown against tampering that included addition or multiplication (but not both simultaneously). In fact, RKA-security for PRFs as defined in [BC10a] is different from BiTR when applied to PRFs. A BiTR PRF is not necessarily RKA-secure since the BiTR simulator is allowed to take some liberties that would violate key independence under tampering as required by RKA-security. We do not pursue these relationships further here formally as it is our intention to capture BiTR in a weakest possible sense and investigate how it captures naturally in a simulation-based fashion the concept of tamper resilience for any cryptographic primitive.

2 BiTR Definitions

Ideal functionalities. Katz [Kat07] modeled usage of a tamper-proof hardware token as an ideal functionality $\mathcal{F}_{wrap}$ in the UC framework. Here, we slightly modify the functionality so that it is parameterized by an interactive Turing machine (ITM) M for a two-party protocol[2] (see Fig. 1). The modification does not change the essence of the wrapper functionality; it merely binds honest parties to the use of a specific embedded program. Corrupted parties may embed an arbitrary program in the token by invoking Forge.

We also define a new functionality $\mathcal{F}_{twrap}$ similar to $\mathcal{F}_{wrap}$ but with tampering allowed. Let $\mathcal{T}$ be a collection of (randomized) functions. Let $\psi = (E, D)$ be an encoding scheme[3]. The essential difference between $\mathcal{F}_{twrap}$ and $\mathcal{F}_{wrap}$ is the ability of the adversary to tamper with the internal state of the hardware token — a function drawn from $\mathcal{T}$ is applied on the internal state of the hardware token. This (weaker) ideal functionality notion is fundamental for the definition of BiTR that comes next.

BiTR Protocols. We define a security notion for a protocol M, called Built-in Tamper Resilience (BiTR), which essentially requires that $\mathcal{F}_{twrap}(M)$ is interchangeable with $\mathcal{F}_{wrap}(M)$. We adopt the notations in the UC framework given by Canetti [Can01].

Definition 1 (BiTR protocol) *The protocol M is $(\mathcal{T}, \psi)$-*BiTR *if for any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that for any non-uniform PPT $\mathcal{Z}$,*

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}} \ ,$$

where $\approx$ denotes computational indistinguishability.

In case $\psi = (\mathsf{id}, \mathsf{id})$ (i.e., identity functions), we simply write $\mathcal{T}$-*BiTR*. Note that this definition is given through the ideal model, which implies (by the standard UC theorem) that whenever a tamper-proof token wrapping M can be used, it can be replaced by a $\mathcal{T}$-tamperable token wrapping M.[4] As a trivial example, every protocol is $\{\mathsf{id}\}$-BiTR.

[2] We will interchangeably use protocols and ITMs, if there is no confusion.

[3] We will sometimes omit ψ from $\mathcal{F}_{twrap}$ when it is obvious from the context.

[4] One could also consider a definition that requires this in the context of a *specific* UC-protocol. We believe our stronger definition, which holds for *any* UC-protocol using a token with M, is the right definition for built-in tamper resilience.

$\mathcal{F}_{wrap}(M)$ is parameterized by a polynomial p and a security parameter k. $\mathcal{F}_{wrap}$ proceeds as follows:

Create: Upon receiving $\langle \mathsf{Create}, sid, P, P', msg \rangle$ from party P:

1. Let $msg' = (\mathsf{Initialize}, msg)$. Run $M(msg')$ for at most $p(k)$ steps.
2. Let *out* be the response of M (set *out* to $\perp$ if M does not respond). Let s' be the updated state of M.
3. Send $\langle \mathsf{Initialized}, sid, P', out \rangle$ to P, and $\langle \mathsf{Create}, sid, P, P', 1^{|s'|} \rangle$ to P' and the adversary.
4. If there is no record $(P, P', *, *)$, then store (P, P', M, s').

Forge: Upon receiving $\langle \mathsf{Forge}, sid, P, P', M', s \rangle$ from the adversary, if P is not corrupted, do nothing. Otherwise do:

1. Send $\langle \mathsf{Create}, sid, P, P', 1^{|s|} \rangle$ to P'.
2. If there is no record $(P, P', *, *)$, then store (P, P', M', s).

Run: Upon receiving $\langle \mathsf{Run}, sid, P, msg \rangle$ from party P', find a record (P, P', K, s). If there is no such record, do nothing. Otherwise, do:

1. Run $K(msg; s)$ for at most $p(k)$ steps.
2. Let *out* be the response of K (set *out* to $\perp$ if K does not respond). Let s' be the updated state of K. Send (sid, P, out) to P'.
3. Update the record with (P, P', K, s').

$\mathcal{F}_{twrap}(M, \mathcal{T}, \psi)$, also parameterized by p and k (and $\psi = (E, D)$ is an encoding scheme), proceeds as follows

Create: As in $\mathcal{F}_{wrap}(M)$ with the only change that state s' is stored as $E(s')$ in memory.

Forge: As in $\mathcal{F}_{wrap}(M)$.

Run: Upon receiving $\langle \mathsf{Run}, sid, P, msg \rangle$ from party P', find a record $(P, P', K, \tilde{s})$. If there is no such record, do nothing. Otherwise, do:

1. (Tampering) if P' is corrupted and a record $\langle sid, P, P', \tau \rangle$ exists set $\tilde{s} = \tau(\tilde{s})$.
2. (Decoding) If P is corrupted, set $s = \tilde{s}$; otherwise, set $s = D(\tilde{s})$. If $s = \perp$, send $(sid, P, \perp)$ to P' and stop.
3. Run $K(msg; s)$ for at most $p(k)$ steps.
4. Let *out* be the response of K (set *out* to $\perp$ if K does not respond). Let s' be the updated state of K. Send (sid, P, out) to P'.
5. (Encoding) If P is corrupted, set $\tilde{s} = s'$; otherwise set $\tilde{s} = E(s')$.
6. Update the record with $(P, P', K, \tilde{s})$.

Tamper: Upon receiving $\langle \mathsf{Tamper}, sid, P, P', \tau \rangle$ from the adversary $\mathcal{A}$, if P' is not corrupted or $\tau \notin \mathcal{T}$, do nothing. Otherwise make a record (sid, P, P', τ) (erasing any previous record of the same form).

Figure 1: Ideal functionalities $\mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap}(M, \mathcal{T}, \psi)$

We note that the above definition is intended to capture in the weakest possible sense the fact that a protocol is tamper resilient within an arbitrary environment. A feature of the definition is that there is no restriction in the way the simulator accesses the underlying primitive (as long as no tampering is allowed). This enables, e.g., a signature to be called BiTR even if simulating tampered signatures requires untampered signatures on different chosen messages, or even on a larger number of chosen messages. We believe that this is the correct requirement for the definition to capture that "if the underlying primitive is secure without tampering, it is secure also with tampering" (in the signature example, security is unforgeability against any polynomial time chosen message attack). Nonetheless, it can be arguably even better to achieve BiTR security through a "tighter" simulation, where the BiTR simulator is somehow restricted to behave in a manner that is closer to the way $\mathcal{A}$ operates (except for tampering of course) or possibly even more restricted. For instance, one may restrict the number of times the token is accessed by the simulator to be upper bounded by the number of times $\mathcal{A}$ accesses the token. In fact all our positive results do satisfy this desired additional tighter simulation property. Taking this logic even further, one may even require that once tampering occurs the BiTR simulator can complete the simulation without accessing the token at all — effectively suggesting that tampering trivializes the token and makes it entirely simulatable (and that would be akin to related-key attack security). We believe that the ability of BiTR to be readily extended to capture such more powerful scenarios highlights the robustness of our notion and, even though these scenarios are not further pursued here, the present work provides the right basis for such upcoming investigations.

2.1 Composition of BiTR ITMs

It is natural to ask if a modular design approach applies to BiTR protocols. To investigate this question we need first to consider how to define the BiTR property in a setting where protocols are allowed to call subroutines.

Consider an ITM M_2 and another ITM M_1 that calls M_2 as a subroutine. We denote by $(M_1; M_2)$ the compound ITM. The internal state of $(M_1; M_2)$ is represented by the concatenation of the two states $s_1||s_2$ where s_1 and s_2 are the states of M_1 and M_2 at a certain moment of the runtime respectively. Let $\mathcal{F}_{twrap}(M_1; M_2, \mathcal{T}_1 \times \mathcal{T}_2, \psi_1 \times \psi_2)$ denote an ideal functionality that permits tampering with functions from $\mathcal{T}_1$ for the state of M_1 and from $\mathcal{T}_2$ for the state of M_2 while the states are encoded with ψ_1 and ψ_2 respectively. We can also consider a sequence of ITMs that call each other successively $\overline{M} = (M_1; \ldots; M_n)$. We next generalize the BiTR notion for an ITM M_i employed in the context of $\overline{M}$ in a straightforward manner[5].

Definition 2 (modular BiTR protocol) *Given $\overline{M} = (M_1; \ldots; M_n)$, $\overline{\mathcal{T}} = \mathcal{T}_1 \times \ldots \times \mathcal{T}_n$, and $\overline{\psi} = \psi_1 \times \ldots \times \psi_n$, for some $i \in [n]$, we say that M_i is modular-$(\mathcal{T}_i, \psi_i)$-BiTR with respect to $\overline{M}, \overline{\mathcal{T}}$ and $\overline{\psi}$ if for any PPT $\mathcal{A}$ there exists a PPT $\mathcal{S}$ such that for any non-uniform PPT $\mathcal{Z}$,*

$$\text{IDEAL}_{\mathcal{F}_{twrap}(\overline{M},\overline{\mathcal{T}_i},\overline{\psi}),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{twrap}(\overline{M},\overline{\mathcal{T}_{i+1}},\overline{\psi}),\mathcal{S},\mathcal{Z}},$$

where $\overline{\mathcal{T}_i} = \{\mathsf{id}\} \times \ldots \times \{\mathsf{id}\} \times \mathcal{T}_i \times \ldots \times \mathcal{T}_n$.

Roughly speaking, this definition requires that whatever the adversary can do by tampering M_i with $\mathcal{T}_i$ (on the left-hand side) should be also done without (on the right-hand side) in the context

[5]We also study the BiTR notion of an ITM M_i when it is universally composed. See later in this section.

of $\overline{M}, \overline{\mathcal{T}}, \overline{\psi}$. For simplicity, if $\overline{M}, \overline{\mathcal{T}}, \overline{\psi}$ are clear from the context, we will omit a reference to it and call an ITM M_i simply modular-$(\mathcal{T}_i, \psi_i)$-BiTR.

The composition theorem below confirms that each ITM being modular BiTR is a necessary and sufficient condition for the overall compound ITM being BiTR.

Theorem 1 (BiTR Composition Theorem) *Consider protocols $M_1, \ldots, M_n$ with $\overline{M} = (M_1, \ldots, M_n)$ and $\mathcal{T} = \mathcal{T}_1 \times \ldots \times \mathcal{T}_n$, and $\psi = \psi_1 \times \ldots \times \psi_n$. It holds that M_i is modular-$(\mathcal{T}_i, \psi_i)$-BiTR for $i = 1, \ldots, n$, with respect to $\overline{M}, \overline{\mathcal{T}}, \overline{\psi}$ if and only if $(M_1; \ldots; M_n)$ is $(\mathcal{T}, \psi)$-BiTR.*

Proof: We need to show that for any $\mathcal{A}$ there is an $\mathcal{S}$ for which it holds that for any $\mathcal{Z}$

$$\text{IDEAL}_{\mathcal{F}_{twrap}(\overline{M},\overline{\mathcal{T}},\overline{\psi}),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(\overline{M}),\mathcal{S},\mathcal{Z}}, \tag{1}$$

First, due to the fact that M_1 is modular BiTR, we obtain that there is an $\mathcal{S}_1$ for which it holds that:

$$\text{IDEAL}_{\mathcal{F}_{twrap}(\overline{M},\overline{\mathcal{T}},\overline{\psi}),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{twrap}(\overline{M},\overline{\mathcal{T}}_2,\overline{\psi}),\mathcal{S}_1,\mathcal{Z}}, \tag{2}$$

Observe that we can advance to the next step by simply applying modular BiTR property again for M_2 and so on successively. This would complete the forward direction of the theorem's statement.

The reverse direction is simple given that all required simulators for showing that M_i is modular BiTR can be instantiated by the simulator $\mathcal{S}$ that is guaranteed to exist since $(M_1; \ldots; M_n)$ is BiTR. ■

While the proof of the composition theorem above is straightforward itdoes highlight a strategy for composing BiTR protocols provided that they are proven to be modular-BiTR with respect to each other. It should be noted that the theorem does not provide the benefits of UC protocol composition that enables arguing about the security of the protocols individually and then arbitrarily composing them. We refer to this issue as the context-sensitivity of the modular BiTR property and we investigate it below. By showing a negative result, wedemonstrate thatit is unavoidable in the context of BiTR security.

Context Sensitivity of Modular-BiTR Security. The modular-BiTR definition is context-sensitive; an ITM may be modular BiTR in some contexts but not in others, in particular depending on the overall compound token $\overline{M}$. This naturally begs a question whether there is a modular-BiTR ITM that is insensitive to the context. In this way, akin to a universally composable protocol, a universally BiTR ITM could be used modularly together with any other ITM and still retain its BiTR property. To capture this we formalize universal-BiTR security below, as well as a weaker variant of it that is called *universal-BiTR parent* which applies only to ITMs used as the parent in a sequence of ITMs.

Definition 3 (universal BiTR) *If an ITM M is modular-$(\mathcal{T}, \psi)$-BiTR with respect to any possible $\overline{M}, \overline{\mathcal{T}}, \overline{\psi}$ then we call M universal-$(\mathcal{T}, \psi)$-BiTR. If M is modular-$(\mathcal{T}, \psi)$-BiTR whenever M is used as the parent ITM then we call it universal-$(\mathcal{T}, \psi)$-BiTR parent.*

Not very surprisingly (and in a parallel to the case of UC protocols) this property is very difficult to achieve. In fact, we show that if one-way functions exist then *non-trivial* universal-BiTR ITMs

do not exist. We first define non-triviality: an ITM M will be called non-trivial if the set of its states can be partitioned into at least two sets S_0, S_1 and there exists a set of inputs A that produce distinct outputs depending when the ITM M is called and its internal state belongs to S_0 or S_1. We call the pair of sets *a state partition for* M and the set A *the distinguishing input-set*. Note that if an ITM is trivial then for any partition of the set of states S_0, S_1 and any set of inputs A, the calling of the ITM M on A produces identical output. This effectively means that the ITM M does not utilize its internal state at all and obviously is BiTR by default. Regarding non-trivial ITMs we next prove that they cannot be $(\mathcal{T}, \psi)$-BiTR for any tampering function τ that switches the state between the two sets S_0, S_1, i.e., $\tau(S_0) \subseteq S_1$, $\tau(S_1) \subseteq S_0$. We call such tampering function *state-switching* for the ITM M. If an encoding ψ is involved, we call τ state-switching for the encoding ψ. We are now ready to prove our negative result.

Theorem 2 *Assuming one-way functions exist, there is no non-trivial universal-$(\mathcal{T}, \psi)$-BiTR ITM M such that $\mathcal{T}$ contains a state-switching function for M and the encoding ψ.*

Proof: Consider a protocol M' that initializes M as well as produces a (vk, sk) pair for a signature scheme. M' ignores any input it is given and always calls M with an input that belongs to A, collects the output z and then returns (z, σ) where $\sigma = Sign(sk, z)$.

Now consider any simulator that tries to simulate a tampering attack against M. In the world where tampering is allowed, after the tampering with the function τ takes place the adversary possesses a signature $\sigma' = Sign(sk, z')$ where z' is the output of M but after the internal state has been switched. The simulator on the other hand can only obtain signatures of the form $(z, Sign(sk, z))$ where $z \neq z'$. Given that M' ignores its input there is no freedom for the simulator to obtain any other signature and as a result under the security of the underlying signature no simulator can succeed in simulating the world where tampering takes place (any successful simulator would amount to a forgery against $Sign(\cdot)$). ■

Roughly speaking, the theorem holds since a parent ITM M_1 calling M_2 can make the message exchanges between them "non-malleable" by outputting a signature on these messages. In this context, no non-trivial M_2 can be modular-BiTR, and thus M_2 is not universal-BiTR. We note that the above theorem is quite final for the case of universal BiTR ITMs. It leaves only the possibility of proving the universal-BiTR property for trivial ITMs (that by default satisfy the notion) or for sets of functions that are not state-switching, i.e., essentially they do not affect the output of M and therefore inconsequential. This state of affairs is not foreign to properties that are supposed to universally compose. Indeed, in the case of UC-security large classes of functionalities are not UC-realizable [CKL03]. To counter this issue, in the UC-setting one may seek setup assumptions to alleviate this problem, but in our setting setup assumptions are to be avoided. For this reason, proving the modular-BiTR property within a given context is preferable.

On the other hand, the universal-BiTR parent property turns out to be feasible, and thus this leaves a context insensitive property to be utilized for modular design of BiTR protocols. We in fact take advantage of this, and jumping ahead, the parent ITM in the compound ITM used to achieve general UC-secure MPC in Section 4 satisfies this property and can be composed with any child ITM.

3 Affine BiTR Protocols without State Encoding

In this section, we show two protocols (for identification and signatures, respectively) that are BiTR against certain tampering functions, without using any modification or encoding. Specifically, we consider a tampering adversary that can modify the state of the hardware with *affine functions*. Assuming the state of the hardware is represented by variables of $\mathbb{Z}_q$ for some prime q, the adversary can choose a tampering $f_{a,b}$ on a variable v, which will change v into $f_{a,b}(v) = av + b \bmod q$. Let $\mathcal{T}_{\mathsf{aff}} = \{f_{a,b} \mid a \in \mathbb{Z}_q^*, b \in \mathbb{Z}_q\}$ and $\mathcal{T}_{\mathsf{aff}}^2 = \mathcal{T}_{\mathsf{aff}} \times \mathcal{T}_{\mathsf{aff}}$.

Schnorr Identification [Sch91]. The Schnorr identification is a two-round two-party protocol between a prover and a verifier. The common input is $y = g^x$, where g is a generator of a cyclic group of size q, and the prover's auxiliary input is $x \in \mathbb{Z}_q$. The protocol proceeds as follows:

1. The prover picks a random $t \in \mathbb{Z}_q$ and sends $z = g^t$ to the verifier.
2. The verifier picks a random $c \in \mathbb{Z}_q$ and sends c to the prover, which in turn computes $s = cx + t \bmod q$ and sends s to the verifier. The verifier checks if $zy^c = g^s$.

We consider the ITM M_{sch} on the prover side wrapped as a hardware token (see Figure 2). This ITM is BiTR against affine functions. To see why it is BiTR, suppose that the adversary tampers with the state changing x into $ax + b$ for some a and b. In the second round, the BiTR simulator — given c, from the adversary, that is supposed to go to $\mathcal{F}_{twrap}(M_{sch}; \mathcal{T}_{\mathsf{aff}})$ — has to find out an appropriate c' going to $\mathcal{F}_{wrap}(M_{sch})$ such that the simulator, on receiving $s' = c'x + t$ from $\mathcal{F}_{wrap}(M_{sch})$, can output $c(ax+b)+t$ that would come from $\mathcal{F}_{twrap}(M_{sch}; \mathcal{T}_{\mathsf{aff}})$. In summary, given (a, b, c, s'), but not x or t, the simulator has to generate a correct output by controlling c'. It can do so by choosing $c' = ac$ and outputting $s' + cb$. Note that $s' + cb = c(ax + b) + t$.

Theorem 3 *The ITM M_{sch} in Fig. 2 is $\mathcal{T}_{\mathsf{aff}}^2$-BiTR without any encoding.*

Proof: Let M be an ITM as described above and $\mathcal{T} = \mathcal{T}_{\mathsf{aff}}^2$. We show that for any non-uniform PPT $\mathcal{Z}$ and any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}}.$$

Let (P, P') be the two party concerned with the token execution; one party generates the token and the other executes it. Handling the case in which no party is corrupted and the case in which both

M_{sch}: The description of a cyclic group of size q, including a generator g, is embedded in the program as a public parameter. The state is $(x, t) \in \mathbb{Z}_q^2$.

Initialization

- Upon receiving a message ($\mathsf{Initialize}$), choose $x, t \in_R \mathbb{Z}_q$ and output g^x.

Message Handling

- Upon receiving a message ($\mathsf{Prove}, \perp$), output $z = g^t$.
- Upon receiving a message (Prove, c), compute $s = cx + t \bmod q$, pick $t \in_R \mathbb{Z}_q$, and output s.

Figure 2: ITM for Schnorr Identification M_{sch}

parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted.

Fix $\mathcal{Z}$ and $\mathcal{A}$. For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(M, \mathcal{T})$. In order to keep the history of tamperings, $\mathcal{S}$ maintains two functions f_x and f_t, which are initialized with identity functions. The simulator $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.
- Upon receiving Create or Forge message, $\mathcal{S}$ just relays the message exchanges between $\mathcal{F}_{wrap}$ and P'.
- Upon receiving $\langle \mathsf{Tamper}, sid, P, P', (f_{a_x,b_x}, f_{a_t,b_t}) \rangle$ from $\mathcal{A}$, Set $f_x = f_x \circ f_{a_x,b_x}$ and $f_t = f_t \circ f_{a_t,b_t}$.
- Upon receiving $\langle \mathsf{Run}, sid, P, msg \rangle$ from $\mathcal{A}$ on behalf of P':

  If $msg = (\mathsf{Prove}, \perp)$, $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle \mathsf{Run}, sid, P, msg \rangle$. Let $\langle sid, P, z \rangle$ be the output from $\mathcal{F}_{wrap}$. $\mathcal{S}$ forwards the output $\mathcal{A}$ and sets f_t to the identity function.

  If $msg = (\mathsf{Prove}, c)$, let w and v (resp., a and b) be the coefficients of f_x (resp., f_t), that is, $f_x : z \mapsto wz + v$ and $f_t : z \mapsto az + b$. $\mathcal{S}$ computes $c' = wc/a$ and calls $\mathcal{F}_{wrap}$ with $\langle Run, sid, P, (Prove, c') \rangle$. Let $\langle sid, P, s' \rangle$ be the output from $\mathcal{F}_{wrap}$. $\mathcal{S}$ computes $s = as' + cv + b$ and sends $\langle sid, P, s \rangle$ to $\mathcal{A}$. Set f_t to the identity function.

Note that

$$\begin{aligned} s &= as' + cv + b = a(c'x + t) + cv + b = a(wcx/a + t) + cv + b \\ &= wcx + at + cv + b = (wx + v)c + (at + b). \end{aligned}$$

Therefore, the above simulation is perfect. ■

Signature Scheme due to Okamoto [Oka06]. The digital signature scheme of Okamoto [Oka06] was employed in the context of designing blind signatures. Here we show that it is BiTR against affine functions. We give a brief description next. Let $(\mathbb{G}_1, \mathbb{G}_2)$ be a bilinear group as follows: (1) $\mathbb{G}_1$ and $\mathbb{G}_2$ are two cyclic groups of prime order q possibly with $\mathbb{G}_1 = \mathbb{G}_2$; (2) h_1 and h_2 are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$ respectively; (3) ψ is an isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$ such that $\psi(h_2) = h_1$; (4) e is a non-degenerate bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ where $|\mathbb{G}_T| = p$, $\forall u \in \mathbb{G}_1 \ \forall v \in \mathbb{G}_2 \ \forall a, b \in \mathbb{Z} : e(u^a, u^b) = e(u, v)^{ab}$.

The signature scheme below is secure against a chosen message attack under the Strong Diffie-Hellman assumption [Oka06].

- Key Generation: Randomly select generators $g_2, u_2, v_2 \in G_2$ and compute $g_1 = \psi(g_2), u_1 = \psi(u_2)$, and $v_1 = \psi(v_2)$. Choose a random $x \in \mathbb{Z}_q^*$ and compute $w_2 = g_2^x$. Verification key is $(g_1, g_2, w_2, u_2, v_2)$. Signing key is x.
- Signature of a message $m \in \mathbb{Z}_q^*$: Choose random $r, s \in \mathbb{Z}_q^*$. The signature is (σ, r, s) where $\sigma = (g_1^m u_1 v_1^s)^{1/(x+r)}$ and $x + r \neq 0 \pmod q$.
- Verification of (m, σ, r, s): Check that $m, r, s, \in \mathbb{Z}_q^*$, $\sigma \in \mathbb{G}_1$, $\sigma \neq 1$, and $e(\sigma, w_2 g_2^r) = e(g_1, g_2^m u_2 v_2^s)$.

M_{oka}: The description of $\mathbb{G}_1$, $\mathbb{G}_2$, g_2, u_2, v_2, and a collision-resistant hashing function $H : \{0,1\}^n \to \mathbb{Z}_q^*$ are embedded in the program as a public parameter. The state is $x \in \mathbb{Z}_q$.

Initialization

- Upon receiving a message (Initialize), choose $x \in_R \mathbb{Z}_q$, and $g_2, u_2, v_2 \in_R G_2$ and output (g_2, w_2, u_2, v_2).

Message Handling

- Upon receiving a message (Sign, m), Choose random $r, s \in \mathbb{Z}_q^*$ such that $x + r \neq 0 \pmod q$. Compute $\sigma = (g_1^{H(m)} u_1 v_1^s)^{1/(x+r)}$ and output (σ, r, s).

Figure 3: Okamoto signature M_{oka}

The signature token is described in Fig. 3. Similarly to the ITM for Schnorr signature scheme, this token can be shown to be BiTR against affine functions.

Theorem 4 *ITM M_{oka} in Fig. 3 is $\mathcal{T}_{\mathsf{aff}}$-BiTR.*

Proof: Let $M = M_{oka}$ and $\mathcal{T} = \mathcal{T}_{\mathsf{aff}}$. We show that for any non-uniform PPT $\mathcal{Z}$ and any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}}.$$

Let (P, P') be the two party concerned with the token execution; one party generates the token and the other executes it. Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted. Fix $\mathcal{Z}$ and $\mathcal{A}$. For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(M, \mathcal{T})$. In order to keep the history of tamperings, $\mathcal{S}$ maintains a function f, which is initialized with the identity function. Wlog, assume the message to be signed is in $\mathbb{Z}_q^*$. The simulator $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.
- Upon receiving Create or Forge message, $\mathcal{S}$ just relays the message exchanges between $\mathcal{F}_{wrap}$ and P'.
- Upon receiving $\langle \mathsf{Tamper}, sid, P, P', f' \rangle$ from $\mathcal{A}$, set $f = f \circ f'$.
- Upon receiving $\langle \mathsf{Run}, sid, P, msg \rangle$ from $\mathcal{A}$ on behalf of P':

  Let $msg = (\mathsf{Sign}, m)$. $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle \mathsf{Run}, sid, P, msg \rangle$. Let $\langle sid, P, (\sigma', r', s') \rangle$ be the output from $\mathcal{F}_{wrap}$. Let a and b be the coefficients of f, (i.e., $f : z \mapsto az + b$). $\mathcal{S}$ computes $\sigma = \sigma'^{1/a}$, $r = ar' - b \bmod q$ and $s = s'$, and sends $\langle sid, P, (\sigma, r, s) \rangle$ to $\mathcal{A}$.

Note that

$$\sigma = \sigma'^{1/a} = (g_1^m u_1 v_1^{s'})^{1/a(x+r')} = (g_1^m u_1 v_1^s)^{1/(ax+r'a)} = (g_1^m u_1 v_1^s)^{1/((ax+b)+r)}$$

and (r, s) is uniformly distributed in $(\mathbb{Z}_q^*)^2$. Therefore the distribution of (σ, r, s) from $\mathcal{S}$ is identical to the distribution from $\mathcal{F}_{twrap}$. ■

Commitment Phase:

1. Each of the sender and the receiver calls $\mathcal{F}_{wrap}(M)$ with a Create message.
2. Each party executes the procedure dual-mode parameter generation with the $\mathcal{F}_{wrap}(M)$. Let pS be the parameter the receiver obtained, and pR be one the sender obtained. The parameters pR and pS are exchanged.
3. The sender commits to a message m by sending $\langle C_1, C_2, \pi \rangle$, where C_1 is a commitment to m based on the parameter pS, C_2 is a statistically-binding commitment to m, and π is WI proof that (1) C_1 and C_2 commits to the same message, or (2) pR was generated in the extraction mode.

Opening Phase:

1. The sender reveals $\langle m, \pi' \rangle$, where m is the committed message, π' is WI proof that (1) C_2 commits to m, or (2) pR was generated in the extraction mode.

Figure 4: A UC Commitment that uses a DPG-commitment scheme Π with protocol M in the $\mathcal{F}_{wrap}(M)$-hybrid model.

4 UC Secure Computation from Tamperable Tokens

In this section we examine the problem of achieving UC-secure computation relying on tamperable (rather than tamper-proof) tokens. Our starting point is the result of Katz [Kat07], obtaining a UC commitment scheme (and general UC-secure computation) in the $\mathcal{F}_{wrap}(M)$-hybrid for an ITM M, which unfortunately, is not BiTR. However, we managed to change M so that the modified ITM M' is BiTR against affine functions, thus obtaining a UC commitment in the $\mathcal{F}_{twrap}(M')$-hybrid. Along the way, we present a generalization of Katz's scheme for building commitment schemes which we call commitments with dual-mode parameter generation. Finally, we examine the OTM (one-time memory) token that was introduced by [GKR08] and used by [GIS$^+$10] to achieve unconditional UC-secure computation. We characterize the BiTR properties of this token.

4.1 Katz's Commitment Scheme and its Generalization.

Intuitively, the UC-secure commitment scheme given by Katz [Kat07] uses the tamper-proof hardware token to give the simulator the advantage over the adversary to force the commitment scheme to become extractable (in case the sender is corrupted) or equivocal (in case the receiver is corrupted). In spirit, this idea can be traced to mixed commitment schemes introduced in [DN02], although the two results differ greatly in techniques.

We abstract the approach of [Kat07] to build UC commitments in Fig. 4. The UC commitment scheme is based on a primitive that we call commitment with dual-mode parameter generation (DPG-commitment for short).

A DPG-commitment is a commitment scheme whose parameter is generated by an interactive protocol M that is wrapped in a hardware token. Formally we define the following:

Definition 4 (DPG-Commitment scheme) *A commitment scheme* $\Pi = (Com, Decom)$ *that*

is parameterized by p, has a dual mode parameter generation (DPG-commitment) if there are ITMs M and P that form a two party protocol $\langle P, M\rangle$ and have the following properties:

- *(Normal mode) For any PPT P^*, with overwhelming probability, the output of $\langle P^*, M\rangle$ satisfies that if it is not $\perp$ then it contains a parameter p over which the commitment scheme Π is unconditionally hiding.*
- *(Extraction mode) For any M^* with the same I/O as M, there is a PPT $\mathcal{S}$ that returns (p, t) such that the commitment scheme Π with the parameter p is a trapdoor commitment scheme with trapdoor t and the parameter generated by $\mathcal{S}$ is computationally indistinguishable from the parameter generated by $\langle P, M^*\rangle$.*

It is worth noting that DPG-commitments are very different from the mixed commitments of [DN02]. For one thing, contrary to mixed commitments, DPG-commitments do not have equivocal parameters. Moreover, mixed commitments have parameters that with overwhelming probability become extractable based on a trapdoor hidden in the common reference string. In contrast, DPG-commitments become extractable due to the manipulation of the parameter generation protocol M (e.g., the ability of the simulator to rewind it). Now using the same arguments as in [Kat07] it is possible to show that the commitment scheme in figure 4 is a UC-commitment provided that the underlying scheme used for C_1 is a DPG-commitment. We briefly sketch the proof argument. When the sender is corrupted, the simulator has to extract the committed message. This can be done by making pS extractable. Then, given a commitment $\langle C_1, C_2, \pi\rangle$ from the adversary, the simulator can extract the message committed to from C_1 using the trapdoor of pS. When the receiver is corrupted, the simulator can make the commitment equivocal by causing pR to be extractable. Using the trapdoor for pR as witness, the simulator can generate a WI proofs π and π' with respect to the condition (2) and thus open the commitment to an arbitrary message.

We next briefly argue that the construction suggested in [Kat07] amounts to a DPG-commitment scheme. The token operates over a multiplicative cyclic group $\mathbb{G}$ of prime order q defined by a safe prime $p = 2q + 1$. In the first round, a party generates a cyclic group and sends to the token the group description and random elements g and h of the group; then, the token sends back a Pedersen [Ped91] commitment $c = \mathsf{com}(g_1, g_2)$ to random elements g_1, g_2 from the group.[6] In the second final round, the party sends a random h_1, h_2, and then the token opens the commitment c and outputs the signature on $(g, h, \hat{g_1}, \hat{g_2})$ where $\hat{g_1} = g_1 h_1$ and $\hat{g_2} = g_2 h_2$. With parameter $(g, h, \hat{g_1}, \hat{g_2})$, the commitment C_1 to a bit b is defined as $(g^{r_1} h^{r_2}, \hat{g_1}^{r_1} \hat{g_2}^{r_2} g^b)$ for randomly-chosen $r_1, r_2 \in \mathbb{Z}_q$. It is well-known (and easy to check) that if the parameter is a Diffie-Hellman (DH) tuple and $r = \log_g \hat{g_1} = \log_h \hat{g_2}$ is known, then b can be efficiently extracted from the commitment. On the other hand, if it is a random tuple, this commitment scheme is perfectly hiding. Extraction mode is achieved by rewinding the code of a malicious token M^*. Specifically for a given M^*, the simulator $\mathcal{S}$ proceeds by picking a random DH tuple $(g, h, \hat{g_1} = g^t, \hat{g_2} = h^t)$ and running M^* once to reach a successful termination and learn the values g_1, g_2. Subsequently, it rewinds M^* right before the second round and selects $h_1 = \hat{g_1}/g_1$ and $h_2 = \hat{g_2}/g_2$. This will result in the parameter produced by M^* to be equal to the DH tuple, i.e., a parameter that is extractable with trapdoor t.

[6] We use a slightly different notation compared to [Kat07] to unify the presentation with our BiTR token that is shown later. Note also that in order to make the committed message lie in the appropriate space, we use a bijection $\mathcal{X} : \mathbb{G} \to [q]$ to encode an element in $\mathbb{G}$, where $\mathcal{X}(\alpha) = \alpha$ if $\alpha \leq q$ or $p - \alpha$ otherwise. The function $\mathcal{X}$ is a bijection, since $|\mathbb{G}| = q$ and for any $\beta \in [q]$, either β or $p - \beta$ belongs to $\mathbb{G}$ (as a quadratic residue).

Let $\mathbb{G}$ be the cyclic multiplicative group of size q defined by a safe prime $p = 2q + 1$ and g be a generator of $\mathbb{G}$. The description of $\mathbb{G}$ is embedded in the program. The state is $(r_1, r_2, s_1, s_2) \in \mathbb{Z}_q^4$. It uses a signature ITM K as a subprotocol.

Initialization

- Upon receiving a message (Initialize), call K with (Initialize), sets the state to all 0s and output whatever K outputs.

Message Handling

- Upon receiving a message h_0: Check h_0 is a generator of $\mathbb{G}$. If the checking fails, output $\perp$. Otherwise, pick $r_i, s_i \in_R \mathbb{Z}_q$ and compute Pedersen commitments $\mathsf{com}_i = g^{s_i} h_0^{\mathcal{X}(g_i)}$ for $i = 1, 2$, where $g_i = g^{r_i}$ and $\mathcal{X}$ is defined as: $\mathcal{X}(\alpha) = \alpha$ if $\alpha \leq q$, or $p - \alpha$ otherwise. Output $(\mathsf{com}_1, \mathsf{com}_2)$.

- Upon receiving a message (h, h_1, h_2, x_1, x_2): Check $h, h_1, h_2 \in \mathbb{G}$, $x_1, x_2 \in \mathbb{Z}_q^*$. If the checking fails, output $\perp$. Otherwise, let $g_i = g^{r_i}$ and compute $\hat{g}_i = g_i^{x_i} h_i$ for $i = 1, 2$. Call K with $(Sign, (P, P', p, g, h, \hat{g}_1, \hat{g}_2))$ to get a signature σ. Output $(g_1, g_2, s_1, s_2, \sigma)$. Pick $r_i, s_i \in_R \mathbb{Z}_q$ for $i = 1, 2$.

Figure 5: Dual parameter generating ITM M_{dpg} that is universal-BiTR parent.

4.2 UC-Secure Commitment Scheme from a Tamperable Token

It is easy to see that the following result holds using the BiTR security properties.

Corollary 5 *If an ITM M, achieving parameters for DPG-commitment scheme, is $\mathcal{T}$-BiTR, then there exists a UC-secure commitment scheme in the $\mathcal{F}_{twrap}(M, \mathcal{T})$-hybrid model.*

Therefore, if the token used in [Kat07] is $\mathcal{T}_{\mathsf{aff}}$-BiTR, then we obtain a UC-secure commitment scheme in the $\mathcal{F}_{twrap}(M, \mathcal{T}_{\mathsf{aff}})$-hybrid model. Unfortunately, the token is not $\mathcal{T}_{\mathsf{aff}}$-BiTR. We explain the issue below. Recall that in the first round the token sends a commitment to g_1, g_2. Suppose that $g_1 = g^{r_1}$ and $g_2 = g^{r_2}$ and that the values r_1 and r_2 are stored as state in the token after the first round. Suppose in addition that by tampering with an affine function the adversary causes the state to become $(ar_1 + b, r_2)$ for some a and b. Then, in the second round, the simulator — given h_1 and h_2 from the adversary — has to send $\mathcal{F}_{wrap}$ appropriate messages h_1' and h_2 so that it can manipulate the output from $\mathcal{F}_{wrap}$ as if the result is from $\mathcal{F}_{twrap}$. Here the signature on $(g, h, \hat{g_1}, \hat{g_2})$ is a critical obstacle, since the simulator cannot modify it (otherwise, it violates unforgeability of signature schemes). This means that for simulation to be successful it should hold that $\hat{g_1} = g^{ar_1+b} h_1 = g^{r_1} h_1'$, i.e., the simulator should select $h_1' = g^{(a-1)r_1+b} h_1$. Unfortunately, the simulator does not know r_1 when it is supposed to send h_1'.

By slightly changing the token above, however, we manage to obtain a DPG-achieving ITM M_{dpg} that is BiTR against affine tampering functions. Its description is given in Fig. 5. First, we show M_{dpg} achieves parameters for DPG-commitment. Roughly speaking, the protocol in the normal mode generates a random tuple $(g, h, \hat{g}_1, \hat{g}_2)$, by multiplying random numbers g_1 and g_2 (from M_{dpg}) and random numbers h_1 and h_2 (from the party). Therefore, the probability that the tuple $(g, h, \hat{g}_1, \hat{g}_2)$ is a DH tuple is negligible since $\hat{g}_1$ and $\hat{g}_2$ are uniformly distributed. In the extraction mode, however, the simulator emulating $\mathcal{F}_{wrap}$ can rewind the ITM to cause $(g, h, \hat{g}_1, \hat{g}_2)$ to be a DH tuple. Specifically, the simulator picks a random DH tuple $(g, h, \hat{g}_1, \hat{g}_2)$ and, after finding out the values g_1, g_2, rewinds the machine right before the second round and sends $h_i = \hat{g}_i / g_i^{x_i}$ for

$i = 1, 2$. Under the DDH assumption, parameters from the normal mode and from the extraction mode are indistinguishable.

More importantly, M_{dpg} is BiTR against affine tampering functions. To achieve BiTR security, we introduce x_1 and x_2. As before, suppose that the state for g_1 is changed from r_1 to $ar_1 + b$. In the second round, the simulator — given h_1 and x_1 — has to send appropriate h'_1 and x'_1 to $\mathcal{F}_{wrap}$ such that $\hat{g_1} = g^{(ar_1+b)x_1}h_1 = g^{r_1x'_1}h'_1$. This means that $h'_1 = g^z h_1$ where $z = (ar_1x_1 + bx_1 - r_1x'_1)$. The good news is that although the simulator does not know r_1, it does know how to pick x'_1 to satisfy the equation: $x'_1 = ax_1$. The value h'_1 can be computed subsequently from the above equation.

Theorem 6 *The ITM M_{dpg} in Fig. 5 is $\mathcal{T}^4_{\mathsf{aff}}$-BiTR.*

Proof: Let $M = M_{dpg}$. We show that for any non-uniform PPT $\mathcal{Z}$ and any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}}.$$

Let (P, P') be the two party concerned with the token execution; one party generates the token and the other executes it. Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted. Fix $\mathcal{Z}$ and $\mathcal{A}$. For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(M, \mathcal{T})$. In order to keep the history of tamperings, $\mathcal{S}$ maintains functions $\{(f^r_i, f^s_i) \mid i = 1, 2\}$, which are initialized with identity functions. The simulator $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.
- Upon receiving Create or Forge message, $\mathcal{S}$ just relays the message exchanges between $\mathcal{F}_{wrap}$ and P'.
- Upon receiving $\langle \mathsf{Tamper}, sid, P, P', (f_{r_1}, f_{r_2}, f_{s_1}, f_{s_2})\rangle$ from $\mathcal{A}$, set $f^r_i = f^r_i \circ f_{r_i}$ and $f^s_i = f^s_i \circ f_{s_i}$ for $i = 1, 2$.
- Upon receiving $\langle \mathsf{Run}, sid, P, msg\rangle$ from $\mathcal{A}$ on behalf of P':
  - If $msg = h_0$, $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle \mathsf{Run}, sid, P, h_0\rangle$ to get output $\langle sid, P, (\mathsf{com}_1, \mathsf{com}_2)\rangle$. $\mathcal{S}$ sends the output to $\mathcal{A}$ and sets f^r_i and f^s_i to identity functions for $i = 1, 2$.
  - If $msg = (h, h_1, h_2, x_1, x_2)$, let w_i and v_i be the coefficients of f^r_i (i.e., $f^r_i : z \mapsto w_i z + v_i$) for $i = 1, 2$. $\mathcal{S}$ computes

    $$h'_i = h_i \cdot g^{v_i x_i}, \quad x'_i = w_i x_i \bmod q,$$

    for $i = 1, 2$ and sends $(h, h'_1, h'_2, x'_1, x'_2)$ to $\mathcal{F}_{wrap}$. Upon receiving $\langle sid, P, (g'_1, g'_2, s'_1, s'_2, \sigma)\rangle$ from $\mathcal{F}_{wrap}$, $\mathcal{S}$ computes $g_i = (g'_i)^{w_i} \cdot g^{v_i}$ and $s_i = f^s_i(s'_i)$ for $i = 1, 2$ and sends $\langle sid, P, (g_1, g_2, s_1, s_2, \sigma)\rangle$ back to $\mathcal{A}$. Set f^r_i and f^s_i to identity functions for $i = 1, 2$.

Note that for $i = 1, 2$

$$\hat{g_i} = (g_i)^{x_i} h_i = ((g'_1)^{w_i} \cdot g^{v_i})^{x_i} h'_i \cdot g^{-v_i x_i} = (g'_1)^{x'_i} h'_i.$$

Therefore the σ is a valid signature on the message $(P, P', g, h, \hat{g}_1, \hat{g}_2)$. The above simulation is perfect. ■

Furthermore, the way the ITM M_{dpg} uses a signature scheme is simple enough (it simply passes through whatever it receives from the signature token) and we can easily extend the above lemma to prove that M_{dpg} is universal BiTR parent. We also show that the ITM for the Okamoto signature scheme M_{oka} is modular-$\mathcal{T}_{\mathsf{aff}}$-BiTR when used with M_{dpg}.

Lemma 7 *ITM M_{oka} in Fig. 3 is modular-$\mathcal{T}_{\mathsf{aff}}$-BiTR with respect to $(M_{dpg}; M_{oka})$.*

Proof: Let $M_1 = M_{dpg}$, $M_2 = M_{oka}$ and $\mathcal{T}_2 = \mathcal{T}_{\mathsf{aff}}$. To show that M_2 is modular BiTR, we need to show for any $\mathcal{A}$, there is an $\mathcal{S}$ such that for all non-uniform $\mathcal{Z}$ it holds that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M_1;M_2,\{\mathsf{id}\}\times\mathcal{T}_2,\{\mathsf{id}\}\times\{\mathsf{id}\}),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M_1;M_2),\mathcal{S},\mathcal{Z}}.$$

Let (P, P') be the two party concerned with the token execution; one party generates the token and the other executes it. Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted. Fix $\mathcal{Z}$ and $\mathcal{A}$. Let $\mathcal{S}_{oka}$ be the BiTR simulator for M_{oka}.

- $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.
- Upon receiving Create or Forge message, $\mathcal{S}$ just relays the message exchanges between $\mathcal{F}_{wrap}$ and P'.
- Upon receiving Tamper message, it forwards the tampering function to $\mathcal{S}_{oka}$.
- Upon receiving Run message, $\mathcal{S}$ forwards it to $\mathcal{F}_{wrap}$ and gets the message. If the reply has a signature part, $\mathcal{S}$ uses $\mathcal{S}_{oka}$ to get the modified signature with the tampered state.

Note that M_{dpg} just passes through the signature part. Also, note that S_{oka} passes through the input message and change only the resulting signature later. Therefore, the signature in the output of M_{dpg} can be handled by $\mathcal{S}_{oka}$. ■

Applying the composition theorem (Theorem 1) along with Theorem 6 and Lemma 7 to the above scheme, we obtain a BiTR token that gives a UC commitment based on corollary 5.

Corollary 8 *$(M_{dpg}; M_{oka})$ is $\mathcal{T}^5_{\mathsf{aff}}$-BiTR.*

4.3 BiTR One-Time Memory (OTM) Tokens

Following [Kat07], several works based UC-secure computation on tamper-proof hardware tokens, in various settings. In [GIS+10] Goyal et. al. show (among other things) protocols for general UC-secure computation in the $\mathcal{F}_{wrap}$(OTM)-hybrid model. This hardware token implements a single OT execution, and was introduced by Goldwasser, Kalai, and Rothblum [GKR08] in the context of one-time programs. Specifically, OTM consists of two k-bit strings[7] $\{s_0, s_1\}$. Upon receiving an input bit b, it outputs s_b and updates the state to consist of $\perp$ ("self-destruct").

[7]For the [GIS+10] result $k = 1$ (namely two bits) is sufficient.

Theorem 9 *The k-bit OTM protocol is $\mathcal{T}$-BiTR if $\mathcal{T}$ can be written as $\mathcal{T} = (\mathcal{T}_0(s_{j_0}), \mathcal{T}_1(s_{j_1}))$ where $\mathcal{T}_i : \{0,1\}^k \to \{0,1\}^k$ and $j_0, j_1 \in \{0,1\}$. That is, each of $\mathcal{T}_i$ depends only on one of the secrets.*

Proof: Let M be an OTM as described and $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2)$ with j_0 and j_1. We show that for any non-uniform PPT $\mathcal{Z}$ and any PPT $\mathcal{A}$, there exists a PPT $\mathcal{S}$ such that

$$\text{IDEAL}_{\mathcal{F}_{twrap}(M,\mathcal{T},\psi),\mathcal{A},\mathcal{Z}} \approx \text{IDEAL}_{\mathcal{F}_{wrap}(M),\mathcal{S},\mathcal{Z}}.$$

Let (P, P') be the two party concerned with the token execution; one party generates the token and the other executes it. Handling the case in which no party is corrupted and the case in which both parties are corrupted is trivial. Now we consider the case in which only one party is corrupted. Wlog, suppose that P' is corrupted. Fix $\mathcal{Z}$ and $\mathcal{A}$. For convenience, let $\mathcal{F}_{wrap} = \mathcal{F}_{wrap}(M)$ and $\mathcal{F}_{twrap} = \mathcal{F}_{twrap}(M, \mathcal{T})$. The simulator $\mathcal{S}$ proceeds as follows:

- $\mathcal{S}$ forwards all the messages between $\mathcal{A}$ and $\mathcal{Z}$.
- Upon receiving Create or Forge message, $\mathcal{S}$ just relays the message exchanges between $\mathcal{F}_{wrap}$ and P'.
- Upon receiving $\langle \mathsf{Tamper}, sid, P, P', (t_0, j_0, t_1, j_1)\rangle$ from $\mathcal{A}$, $\mathcal{S}$ records (t_0, j_0, t_1, j_1).
- Upon receiving $\langle \mathsf{Run}, sid, P, msg\rangle$ from $\mathcal{A}$ on behalf of P':

  Let $msg = b \in \{0,1\}$. If tampering is not recorded, $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle \mathsf{Run}, sid, P, msg\rangle$ and forwards the output from $\mathcal{F}_{wrap}$ to $\mathcal{A}$. Otherwise, let (t_0, j_0, t_1, j_1) be the recorded tampering. $\mathcal{S}$ calls $\mathcal{F}_{wrap}$ with $\langle \mathsf{Run}, sid, P, j_b\rangle$. Let $\langle sid, P, z\rangle$ be the output from $\mathcal{F}_{wrap}$. $\mathcal{S}$ sends $\langle sid, P, t_b(z)\rangle$ to $\mathcal{A}$.

It is easy to check $t_b(z) = t_b(s_{j_b})$ is equal to the tampered state s'_b.

■

5 BiTR Protocols against General Classes of Tampering Functions

5.1 BiTR Protocols from Non-Malleable Codes

In this section we will see how the BiTR property can be derived by implementing an integrity check in the form of an encoding ψ. A useful tool for this objective is the notion of non-malleable codes [DPW10]. A pair of procedures (E, D) is a non-malleable code with respect to tampering functions $\mathcal{T}$, if there is an algorithm $\mathcal{S}$ that detects whether the state becomes invalid, given only the tampering function t. In particular, $\mathcal{S}$ should satisfy the following properties for all $x \in \{0,1\}^n$ and $t \in \mathcal{T}$:

- If $x = D(t(E(x)))$ (i.e., x stays the same even after applying the tampering t), it holds that $\mathcal{S}(t) = \mathsf{ok}$ with overwhelming probability.
- Otherwise, $\mathcal{S}(t)$ is statistically (or computationally) close to $D(t(E(x)))$.

By encoding the state of a protocol with a non-malleable code it is possible to show the following restatement of Theorem 6.1 of [DPW10] under the BiTR security framework.

Theorem 10 ([DPW10]) *Let $\mathcal{T}$ be a class of tampering functions over $\{0,1\}^m$ and $(E, D, \mathcal{S})$ be a non-malleable code with respect to $\mathcal{T}$, where $E : \{0,1\}^n \to \{0,1\}^m$, $D : \{0,1\}^m \to \{0,1\}^n$ and $\mathcal{S}$ are efficient procedures. Let M be any ITM whose state is of length n. Then M is $(\mathcal{T}, \psi)$-BiTR where $\psi = (E, D)$.*

The above theorem suggests the importance of the problem of constructing non-malleable codes for a given class of tampering functions $\mathcal{T}$. Some positive answers to this difficult question are given in [DPW10] for a class of tampering functions that operate on each one of the bits of the state independently; they also provide a general feasibility result for tampering families of bounded size (with an inefficient construction); an important characteristic of those solutions is relying on the randomness of the encoding. Here we show a different set of positive results by considering the case of *deterministic* non-malleable codes, i.e., the setting where (E, D) are both deterministic functions.

In our result we will utilize a relaxation of non-malleable codes: $(E, D, Predict)$ is called a δ-non-malleable code with distance ϵ if for any $x \in \{0,1\}^n$ and $t \in \mathcal{T}$, the following holds:

(i) $D(E(x)) = x$.

(ii) The probability that $D(t(E(x)))$ is neither x nor $\perp$ is at most δ,[8] and

(iii) $Predict(\cdot)$ outputs either ok or $\perp$; moreover, $|\Pr[D\big(t(E(x))\big) = x] - \Pr[Predict(t) = \mathsf{ok}]| \leq \epsilon$.

It is easy to see that if ϵ and δ are negligible the resulting code is non-malleable: given that δ is negligible, property (ii) suggests that D will return either the correct value or fail, and thus in case it fails, $Predict(\cdot)$ will return $\perp$ with about the same probability due to (iii). We call δ the crossover threshold and ϵ the predictability distance.

Note that a δ-non-malleable code with only a negligible ϵ is not sufficient by itself in simulating tampering attacks since the $Predict(\cdot)$ algorithm can only guess whether the decoding algorithm will succeed (but is not capable of extracting the proper decoding in case a crossover happens). As we show next this is a very useful relaxation that can be taken advantage of when designing deterministic non-malleable codes.

5.2 Constructing Deterministic Non-Malleable Codes

We now consider the problem of constructing a δ-non-malleable code $E : \{0,1\}^n \to \{0,1\}^m$ for a given class of tampering functions and parameters δ, ϵ. We will only consider the case when $\delta > \epsilon$ as the other case is not useful. We note that the construction is inefficient for large m and n, but it becomes efficient for logarithmic values of m, n. Following this we utilize it in the construction of deterministic non-malleable codes.

Graph Theoretic Construction. For a given $t \in \mathcal{T}$ consider the graph G that is defined with vertex set $V = \{0,1\}^m$ with each edge (u_1, u_2) having weight $w_t(u_1, u_2) = \Pr[t(u_1) = u_2]$.[9] Finding a good δ-non-malleable code amounts to finding a partition $S, \overline{S} = V \setminus S$ of G satisfying the following properties that for each $t \in \mathcal{T}$:

[8] The tampering t may change the codeword x into another valid codeword.

[9] In the above description, we assumed the probabilities $\Pr[t(c) = u]$ are known. If they are not known, they can be estimated using standard techniques. In particular, to evaluate the probability of an event A, repeat k independent experiments of A and denote the success ratio of the k experiments as $\hat{p}$. Let X_i be the probability that the i-th execution of the event A is successful. The expected value of $Y = \sum_{i=1}^{k} X_i$ is $k \cdot p$. Using the Chernoff bound it follows that $|\hat{p} - p| \leq 1/N$ with probability $1 - \gamma$ provided that $k = \Omega(N^2 \ln(\gamma^{-1}))$.

(1) For all $u \in S$, it holds that $\sum_{v \in S \setminus \{u\}} w_t(u,v) \leq \delta$.

(2) One of the following is true.

  (a) For all $u \in S$, $\sum_{v \in \overline{S}} w_t(u,v) \geq 1 - \epsilon$.

  (b) For all $u \in S$, $\sum_{v \in \overline{S}} w_t(u,v) \leq \epsilon$.

Given that the partition S meets the condition (2) for all functions in $\mathcal{T}$, we will say that S is a *repeller* (resp., an *attractor*) for a given $t \in \mathcal{T}$ if S satisfies condition (a) (resp., condition (b)) for t. We remark that when t makes S a repeller for it, the condition (1) becomes obsolete since the condition (a) requires the probability of moving from S to $\overline{S}$ to be very high (i.e., this will immediately force the δ crossover bound).

Note that we are interested in the largest such S since this provides best encoding rate, since we can construct a non-malleable code from S as follows:

> Set $\mathcal{C} \subseteq V = \{0,1\}^m$ and $n = \lfloor \log_2 |\mathcal{C}| \rfloor$. The encoding function $E : \{0,1\}^n \to \{0,1\}^m$ is an arbitrary injection from $\{0,1\}^n$ to $\mathcal{C}$. The decoding D is defined as the inverse of E when restricted on $\mathcal{C}$, and $\perp$ everywhere else. On input t, the *Predict* algorithm outputs ok if $\mathcal{C}$ is an attractor for t; otherwise output $\perp$ (i.e., if $\mathcal{C}$ is an repeller for t).

Difficulty of Constructing δ-Non-Malleable Codes. We show that the problem of constructing optimal non-malleable codes is NP-hard and even hard to approximate. In particular, we show a reduction from the maximum independent set problem to this problem. We will only consider the problem of finding the optimal S that satisfies condition (a). Consider an undirected connected graph $G = (V, E)$. Let $M = |V|$. We construct a directed weighted graph $G^* = (V, E^*, w)$ as follows:

> Define $E^* = \{(u,v) : u, v \in V \text{ and } u \neq v\}$. The weight function $w(u,v)$ is defined as follows:
>
> $$w(u,v) = \begin{cases} \frac{1}{M^2} & \text{if } (u,v) \notin E \\ c_u & \text{if } (u,v) \in E \end{cases}$$
>
> where c_u is a fixed value (depending only on u) that makes $\sum_v w(u,v) = 1$. In particular, letting $N_u = |\{v : \ (u,v) \in E\}|$ and $N'_u = |\{v : \ (u,v) \notin E\}|$, the value c_u is computed as $\frac{1}{N_u} \cdot (1 - \frac{N'_u}{M^2})$.

Note that it holds that for any $u \in V$

$$c_u = \frac{1}{M} \cdot \frac{M - N'_u/M}{N_u} > \frac{1}{M} \cdot \frac{M - N'_u}{N_u} = \frac{1}{M} \cdot \frac{N_u + 1}{N_u} > \frac{1}{M}.$$

We next consider any δ-non-malleable code S for the tampering function that is defined by the graph G^* parameterized with $\delta = 1/M$ and $\epsilon = 1/M - 1/M^2$. If the set S is an *attractor* this means that for all $u \in S$, $\sum_{v \in \overline{S}} w(u,v) \leq \epsilon = 1/M - 1/M^2$. Therefore, it holds that for any (u,v) with $u \in S$ and $v \in \overline{S}$, we have $(u,v) \notin E$; otherwise, we will have $\sum_{v \in \overline{S}} w(u,v) \geq c_u > \frac{1}{M}$, which is a contradiction. However, this implies that in the original graph G, any node $u \in S$ is disconnected from any node $v \in \overline{S}$, which is a contradiction to that G is a connected graph.

Thus, we are left necessarily with the case that the δ-non-malleable code S is a *repeller*. In this case, it holds that for all $u \in S$, $\sum_{v \in \overline{S}} w(u,v) \geq 1 - \epsilon$ and as a result $\sum_{v \in S \setminus \{u\}} w(u,v) \leq \epsilon <$

$\delta = 1/M$. Now observe that if $(u, v) \in E$, then in the graph G^*, we have $w(u, v) = c_u > 1/M$. As a result, for any $(u, v) \in E$ with $u \in S$, it must be the case $v \in \bar{S}$. This means that S is an independent set.

Based on the above it follows immediately that finding the maximum independent set reduces to finding the optimal code for a suitably defined tampering function (a single function is sufficient). We conclude the following:

Proposition 11 *The problem of finding an optimal δ-non-malleable code given $\mathcal{T}, \delta, \epsilon$ is NP-hard.*

Heuristic Construction. We next provide a simple heuristic that is guaranteed to produce a code of non-zero rate if such exists. It operates as follows: as before consider the graph G defined by the tampering function $t \in \mathcal{T}$. We consider all pairs of vertices $\{u_1, u_2\}$ and classify them according to whether they are repellers or attractors with parameters δ, ϵ. Note that testing whether these sets are repellers or attractors requires $poly(|V|)$ steps. We perform the same for all tampering functions $t \in \mathcal{T}$ and then consider only those sets that appear in the list of all tampering functions. Finally, we improve the size of such a selected pair by moving vertices from $\overline{S}$ to S provided that the repeller or attractor property is maintained. We note that this approach will enable us to reach a local maximum code nevertheless it is not guaranteed to find an optimal code (something expected in light of the proposition above).

The rate of the constructed code is n/m, while the time-complexity of constructing the code is $2^{\mathcal{O}(n)}|\mathcal{T}|$. The size of the circuit evaluating each one of these functions is respectively $2^n, 2^m, |\mathcal{T}|$. The proof of the following theorem follows easily from the fact that our construction above considers all possible pairs and as such it is guaranteed to find at least one that works for all tampering functions if one exists.

Theorem 12 *Fix any class of functions $\mathcal{T}$. If there exists a code $(E, D, Predict)$ with rate > 0 that is δ-non-malleable w.r.t. $\mathcal{T}$ and distance ϵ, then such a code is produced by the above procedure.*

Despite the weak message-rate preservation achieved in the theorem above, we demonstrate later in this section how the theorem is a sufficient building block for obtaining efficient constructions of deterministic non-malleable codes for a large class of tampering functions.

When Does a Deterministic Non-Malleable Code Exist? The basic idea of the construction above was to search for sets of codewords that are either attractors or repellers in the graphs that are defined by the tampering functions. The necessity of finding such subsets follows from the fact that any set of vertices that is neither an attractor or a repeller will fail to provide a δ-non-malleable code since the $Predict(\cdot)$ function will be impossible to define in this case correctly. Specifically, if the set S is neither a repeller nor an attractor it holds that there is some codeword $u \in S$ and tampering function t for which $\epsilon < \Pr[t(u) \notin S] < 1 - \epsilon$. It follows that when provided t, $Predict(\cdot)$ will fail to give the proper response as required by the specification. As a result a δ-non-malleable code will exist for a family $\mathcal{T}$ provided that there is a set S for which it holds that is either a repeller or attractor for each $t \in \mathcal{T}$.

Explicit Constructions. We next provide two illustrative examples and discuss the existence (and rate) of explicit deterministic non-malleable encodings for them.

Example 1: Set Functions. If $\mathcal{T}$ contains a function t that sets the i-th bit of $u \in \{0, 1\}^m$ to 0, it follows that the code $\mathcal{C}$ we construct must obey that either all codewords have the i-th bit set to 0

or all of them have the bit set to 1 due to the crossover requirement. This means that the inclusion of any bit setting function in $\mathcal{T}$ cuts the size of the code $|\mathcal{C}|$ by half. There is no non-malleable code when the collection $\mathcal{T}$ contains Set functions for every bit position (this is consistent with the impossibility result of [GLM+04] for algorithmic tamper proof security when Set functions are allowed for tampering).

Example 2: Differential Fault Analysis [BDL01]. Consider a single function t which flips each 1-bit to a 0-bit with probability β. Consider a code $\mathcal{C} \subseteq \{0,1\}^m$ for which it holds that all codewords in $\mathcal{C}$ have Hamming distance at least r between each other and $0^m \in \mathcal{C}$. Then it is easy to see that δ, the probability of crossover, is at most β^r. Further, now suppose that t is applied to an arbitrary codeword u in $\mathcal{C}$ other than 0^m. We observe that the number of 1's in u is at least r (otherwise it would have been too close to 0^m). It follows that t will change some of these 1's to 0's, with probability at least $1-(1-\beta)^r$. It follows that we can predict the effect of the application of t with this probability when we restrict to codewords in $\mathcal{C} \setminus \{0^m\}$. In summary, any code $\mathcal{C}$ over $\{0,1\}^m$ with minimum distance r that contains 0^m allows for a β^r-non-malleable code with $(1-\beta)^r$ for t using the code $\mathcal{C} \setminus \{0^m\}$.

We can extend the above to the case when multiple applications of t (say up to a times) are allowed before running the next round of the protocol with the token. Note that a sequence of a applications of t will flip each 1-bit to a 0-bit with probability $\beta + (1-\beta)\beta + \ldots + (1-\beta)^{a-1}\beta = 1-(1-\beta)^a$. The encoding now has crossover $(1-(1-\beta)^a)^r \leq e^{-(1-\beta)^a r}$. Thus, from $e^{-(1-\beta)^a r} \leq \delta$, we obtain $r \geq (1/(1-\beta))^a \ln(1/\delta)$, i.e., when β is bounded away from 1, the minimum distance of the code grows exponentially with a.

Efficient Construction for any $\mathcal{T}$ that is *localized*. Now, we show a simple way to use the (inefficient) construction of the beginning of the section with constant rate and any crossover $\delta < 1/2$, to achieve an efficient construction with negligible crossover (and thus, BiTR security for any protocol M whose state is encoded with the resulting code), when the class contains only functions that can be split into independent tampering of local (i.e., logarithmically small) blocks. Here we consider a tampering class $\mathcal{T}$ of polynomial size. Roughly speaking, the construction is achieved first by applying a Reed-Solomon code to the overall state and then by applying the δ-non-malleable code to the resulting codeword in small blocks. Let $\mathcal{T}^\ell$ denote $\mathcal{T} \times \cdots \times \mathcal{T}$ (with ℓ repetitions).

Theorem 13 *Let k be a security parameter. Let $\mathcal{T}$ be a class of functions over $\{0,1\}^m$ with $m = \mathcal{O}(\log k)$ for which a δ-non-malleable code exists and is efficiently constructible with rate r. Then there is an efficiently constructible deterministic non-malleable code w.r.t. $\mathcal{T}^\ell$ for any rate less than $(1-\delta)r$ provided $\ell/\log \ell = \omega(\log k)$.*

Proof: We first utilize the construction of the beginning of the section to obtain a δ-non-malleable code $E : \{0,1\}^n \to \{0,1\}^m$, $D, Predict(\cdot)$ for which it holds that ϵ is negligible. We note that this is feasible in polynomial time in k.

Using this code, we construct a non-malleable code w.r.t. $\mathcal{T}^\ell$ as follows.

Encoding. We show a construction of an encoding function $E' : \{0,1\}^{nq} \to \{0,1\}^{m\ell}$. The parameter q will be determined later according to the desired rate of the code. Given an input $x \in \{0,1\}^{nq}$, the encoding proceeds as follows:

1. Parse the given input $x \in \{0,1\}^{nq}$ as $(x_1, \ldots, x_q) \in \{0,1\}^{n \times q}$. Apply a Reed-Solomon code

to $(x_1, \ldots, x_q)$ and obtain a codeword $y = (y_1, \ldots, y_{\tilde{\ell}}) \in \{0,1\}^{\tilde{n} \times \tilde{\ell}}$:

$$y_j = \sum_{i=0}^{q-1} (\alpha_j)^i x_{i+1},$$

where $\tilde{n}$ is the multiple of n with $2^{\tilde{n}} > \tilde{\ell}$, and $\alpha_1, \ldots, \alpha_{\tilde{\ell}}$ are distinct elements (such elements exist given that $2^{\tilde{n}} > \tilde{\ell}$).

2. Parse y as $(u_1, \ldots, u_\ell) \in \{0,1\}^{n \times \ell}$ such that $n\ell = \tilde{n}\tilde{\ell}$. For $j = 1, \ldots, \ell$, encode each u_j to $z_j = E(u_j)$. The overall encoding is defined as $(z_1, \ldots, z_\ell)$.

Observe that the rate of this encoding E' is ζr where $\zeta = q/\ell$.

Decoding. To decode a codeword $(z_1, \ldots, z_\ell)$, compute $u_i = D(z_i)$ for $i = 1, \ldots, \ell$, where D is the decoding function of the δ-non-malleable code w.r.t. $\mathcal{T}$. If one of those individual decodings fails then decoding fails. Otherwise, parse $(u_1, \ldots, u_\ell) \in \{0,1\}^{n \times \ell}$ into $(y_1, \ldots, y_{\tilde{\ell}}) \in \{0,1\}^{\tilde{n} \times \tilde{\ell}}$, and check if the points $\{(\alpha_i, y_i)\}_{i=1}^{\tilde{\ell}}$ lie on a polynomial of degree less than q. If this is the case output the polynomial's coefficients $(x_1, \ldots, x_q)$; otherwise the decoding fails.

Now, consider the family of tampering functions $\mathcal{T}^\ell$. We show the crossover parameter of the above construction is $negl(k)$. Given that the action in each coordinate is independent, the probability of crossover is the probability of switching $\ell - q + 1 = (1 - \zeta)\ell + 1$ coordinates. This probability is negligibly small under the conditions that $\delta < 1 - \zeta$ and $\ell / \log \ell = \omega(\log k)$, via Chernoff Bound. Predictability is easily satisfied by checking if the output of Predict for each block is 1. ■

References

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In *CRYPTO*, pages 36–54, 2009.

[AGV09] Adi Akavia, Shafi Goldwasser, and Vinod Vaikuntanathan. Simultaneous hardcore bits and cryptography against memory attacks. In *TCC*, pages 474–495, 2009.

[AHI11] Benny Applebaum, Danny Harnik, and Yuval Ishai. Semantic security under related-key attacks and applications. In *Innovations in Computer Science - ICS 2011*, pages 45–60, 2011.

[AK96] Ross J. Anderson and Markus G. Kuhn. Tamper resistance – a cautionary note. In *The Second USENIX Workshop on Electronic Commerce Proceedings*, pages 1–11, Oakland, California, 18–21 1996.

[BB05] David Brumley and Dan Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

[BC10a] Mihir Bellare and David Cash. Pseudorandom functions and permutations provably secure against related-key attacks. In *CRYPTO*, pages 666–684, 2010.

[BC10b] Nir Bitansky and Ran Canetti. On strong simulation and composable point obfuscation. In *CRYPTO*, pages 520–537, 2010.

[BDL01] Dan Boneh, Richard A. DeMillo, and Richard J. Lipton. On the importance of eliminating errors in cryptographic computations. *J. Cryptology*, 14(2):101–119, 2001.

[BECN+04] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerers apprentice guide to fault attacks. Cryptology ePrint Archive, Report 2004/100, 2004.

[BG10] Zvika Brakerski and Shafi Goldwasser. Circular and leakage resilient public-key encryption under subgroup indistinguishability - (or: Quadratic residuosity strikes back). In *CRYPTO*, pages 1–20, 2010.

[BK03] Mihir Bellare and Tadayoshi Kohno. A theoretical treatment of related-key attacks: Rka-prps, rka-prfs, and applications. In *EUROCRYPT*, pages 491–506, 2003.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *FOCS*, pages 501–510, 2010.

[BS97] Eli Biham and Adi Shamir. Differential fault analysis of secret key cryptosystems. In *CRYPTO*, pages 513–525, 1997.

[Can01] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *FOCS*, pages 136–145, 2001.

[CF01] Ran Canetti and Marc Fischlin. Universally composable commitments. In *CRYPTO*, pages 19–40, 2001.

[CGGM00] Ran Canetti, Oded Goldreich, Shafi Goldwasser, and Silvio Micali. Resettable zero-knowledge (extended abstract). In *STOC*, pages 235–244, 2000.

[CGS08] Nishanth Chandran, Vipul Goyal, and Amit Sahai. New constructions for uc secure computation using tamper-proof hardware. In *EUROCRYPT*, pages 545–562, 2008.

[CKL03] Ran Canetti, Eyal Kushilevitz, and Yehuda Lindell. On the limitations of universally composable two-party computation without set-up assumptions. In *EUROCRYPT*, pages 68–86, 2003.

[DGK+10] Yevgeniy Dodis, Shafi Goldwasser, Yael Tauman Kalai, Chris Peikert, and Vinod Vaikuntanathan. Public-key encryption schemes with auxiliary inputs. In *TCC*, pages 361–381, 2010.

[DKL09] Yevgeniy Dodis, Yael Tauman Kalai, and Shachar Lovett. On cryptography with auxiliary input. In *STOC*, pages 621–630, 2009.

[DKM11] Nico Döttling, Daniel Kraschewski, and Jörn Müller-Quade. Unconditional and composable security using a single stateful tamper-proof hardware token. In *TCC*, pages 164–181, 2011.

[DN02] Ivan Damgård and Jesper Buus Nielsen. Perfect hiding and perfect binding universally composable commitment schemes with constant expansion factor. In *CRYPTO*, pages 581–596, 2002.

[DNW08] Ivan Damgård, Jesper Buus Nielsen, and Daniel Wichs. Isolated proofs of knowledge and isolated zero knowledge. In *EUROCRYPT*, pages 509–526, 2008.

[DP10] Yevgeniy Dodis and Krzysztof Pietrzak. Leakage-resilient pseudorandom functions and side-channel attacks on feistel networks. In *CRYPTO*, pages 21–40, 2010.

[DPW10] Stefan Dziembowski, Krzysztof Pietrzak, and Daniel Wichs. Non-malleable codes. In *ICS*, pages 434–452, 2010.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In *TCC*, pages 343–360, 2010.

[FRR+10] Sebastian Faust, Tal Rabin, Leonid Reyzin, Eran Tromer, and Vinod Vaikuntanathan. Protecting circuits from leakage: the computationally-bounded and noisy cases. In *EUROCRYPT*, pages 135–156, 2010.

[GIMS10] Vipul Goyal, Yuval Ishai, Mohammad Mahmoody, and Amit Sahai. Interactive locking, zero-knowledge pcps, and unconditional cryptography. In *CRYPTO*, pages 173–190, 2010.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. One-time programs. In *CRYPTO*, pages 39–56, 2008.

[GLM+04] Rosario Gennaro, Anna Lysyanskaya, Tal Malkin, Silvio Micali, and Tal Rabin. Algorithmic tamper-proof (atp) security: Theoretical foundations for security against hardware tampering. In *TCC*, pages 258–277, 2004.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In *CRYPTO*, pages 59–79, 2010.

[HL11] Shai Halevi and Huijia Lin. After-the-fact leakage in public-key encryption. In *TCC*, pages 107–124, 2011.

[IPSW06] Yuval Ishai, Manoj Prabhakaran, Amit Sahai, and David Wagner. Private circuits ii: Keeping secrets in tamperable circuits. In *EUROCRYPT*, pages 308–327, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, pages 463–481, 2003.

[JV10] Ali Juma and Yevgeniy Vahlis. Protecting cryptographic keys against continual leakage. In *CRYPTO*, pages 41–58, 2010.

[Kat07] Jonathan Katz. Universally composable multi-party computation using tamper-proof hardware. In *EUROCRYPT*, pages 115–128, 2007.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, pages 388–397, 1999.

[Kol10] Vladimir Kolesnikov. Truly efficient string oblivious transfer using resettable tamper-proof tokens. In *TCC*, pages 327–342, 2010.

[KV09] Jonathan Katz and Vinod Vaikuntanathan. Signature schemes with bounded leakage resilience. In *ASIACRYPT*, pages 703–720, 2009.

[LW10] Allison B. Lewko and Brent Waters. On the insecurity of parallel repetition for leakage resilience. In *FOCS*, pages 521–530, 2010.

[MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In *TCC*, pages 278–296, 2004.

[MS08] Tal Moran and Gil Segev. David and goliath commitments: Uc computation for asymmetric parties using tamper-proof hardware. In *EUROCRYPT*, pages 527–544, 2008.

[MTVY11] Tal Malkin, Isamu Teranishi, Yevgeniy Vahlis, and Moti Yung. Signatures resilient to continual leakage on memory and computation. In *TCC*, pages 89–106, 2011.

[NS09] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. In *CRYPTO*, pages 18–35, 2009.

[Oka06] Tatsuaki Okamoto. Efficient blind and partially blind signatures without random oracles. In *TCC*, pages 80–99, 2006.

[P11] Abhishek Jain 0002 and Krzysztof Pietrzak. Parallel repetition for leakage resilience amplification revisited. In *TCC*, pages 58–69, 2011.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, pages 129–140, 1991.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In *EUROCRYPT*, pages 462–482, 2009.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In *E-SMART*, pages 200–210, 2001.

[SA02] Sergei P. Skorobogatov and Ross J. Anderson. Optical fault induction attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 2–12. Springer, 2002.

[Sch91] Claus-Peter Schnorr. Efficient signature generation by smart cards. *J. Cryptology*, 4(3):161–174, 1991.

[Sko05] Sergei P. Skorobogatov. Semi-invasive attacks – A new approach to hardware security analysis. Technical Report UCAM-CL-TR-630, University of Cambridge, Computer Laboratory, April 2005.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, pages 443–461, 2009.