A version of this article was accepted for publication in the International Journal of Information Security. The full version is available in the IACR eprint archive.

Algorithm Substitution Attacks against Receivers

Marcel Armour · **Bertram Poettering**

Abstract This work describes a class of Algorithm Substitution Attack (ASA) generically targeting the receiver of a communication between two parties. Our work provides a unified framework that applies to any scheme where a secret key is held by the receiver; in particular, message authentication schemes (MACs), authenticated encryption (AEAD) and public key encryption (PKE). Our unified framework brings together prior work targeting MAC schemes (FSE'19) and AEAD schemes (IMACC'19) ; we extend prior work by showing that public key encryption may also be targeted.

ASAs were initially introduced by Bellare, Paterson and Rogaway in light of revelations concerning mass surveillance, as a novel attack class against the confidentiality of encryption schemes. Such an attack replaces one or more of the regular scheme algorithms with a subverted version that aims to reveal information to an adversary (engaged in mass surveillance), while remaining undetected by users. Previous work looking at ASAs against encryption schemes can be divided into two groups. ASAs against PKE schemes target key generation by creating subverted public keys that allow an adversary to recover the secret key. ASAs against symmetric encryption target the encryption algorithm and leak information through a subliminal channel in the ciphertexts. We present a new class of attack that targets the decryption algorithm of an encryption scheme for symmetric encryption and public key encryption, or the verification algorithm for an authentication scheme. We present a generic framework for subverting a cryptographic scheme between a sender and receiver, and show how a decryption oracle allows a subverter to create a subliminal channel which can be used to leak secret keys. We then show that the generic framework can be applied to authenticated encryption with associated data, message authentication schemes, public key encryption and KEM/DEM constructions.

We consider practical considerations and specific conditions that apply for particular schemes, strengthening the generic approach. Furthermore, we show how the hybrid subversion of key generation and decryption algorithms can be used to amplify the effectiveness of our decryption attack. We argue that this attack represents an attractive opportunity for a mass surveillance adversary. Our work serves to refine the ASA model and contributes to a series of papers that raises awareness and understanding about what is possible with ASAs.

Keywords Algorithm Substitution Attacks · Privacy · Mass Surveillance · Cryptography

Marcel Armour
Royal Holloway University of London, Egham, UK
E-mail: marcel.armour.2017@rhul.ac.uk
ORCID: 0000-0002-1231-6120

Bertram Poettering
IBM Research Europe – Zurich, Rüschlikon, Switzerland
E-mail: poe@zurich.ibm.com
ORCID: 0000-0001-6525-5141

1 Introduction

Consider two parties communicating over an untrusted channel (in the presence of an adversary). Desired security properties for this scenario include confidentiality and integrity. Confidentiality means that the adversary is unable to learn anything about the messages sent between the parties. Integrity means that the parties can be sure that the messages have not been tampered with in transit. Both confidentiality and integrity are well studied problems and there are many reliable and provably secure cryptographic solutions. These solutions rely on the assumption that the software or hardware in which they are implemented behaves as expected. However, we know that in the real world this assumption does not necessarily hold. Powerful adversaries have the means to insert unreliability into cryptography via external ("real-world") infrastructure: whether by influencing standards bodies to adopt "backdoored" parameters, in-

serting exploitable errors into software implementations, or compromising supply chains to interfere with hardware. The Snowden revelations showed that this is indeed the case, and that large and powerful adversaries (interested in mass surveillance) have sought to circumvent cryptography. The reader is referred to the survey by Schneier et al. [59], which provides a broad overview of subversion of cryptography, with some useful case studies detailing known subversion attempts.

The idea that an adversary may embed a backdoor or otherwise tamper with the implementation or specification of a cryptographic scheme or primitive predates the Snowden revelations, and was initiated in a line of work by Young and Yung that they named *kleptography* [62,63]. This area of study can be traced back to Simmons' work on *subliminal channels*, e.g. [60], undertaken in the context of nuclear non-proliferation during the Cold War. In the original conception [62], kleptography considered a saboteur who designs a cryptographic algorithm whose outputs are computationally indistinguishable from the outputs of an unmodified trusted algorithm. The saboteur's algorithm should leak private key data through the output of the system, which was achieved using the same principles as Simmons' earlier subliminal channels. Post-Snowden, work in this area was reignited by Bellare, Paterson and Rogaway (BPR) [14], who formalised the study of so-called *algorithm substitution attacks* (ASAs) through the example of symmetric encryption schemes. In abstract terms, the adversary's goal in an ASA is to create a subverted implementation of a scheme that breaks some aspect of security (such as IND-CPA in the case of encryption) while remaining undetected by the user(s).

1.1 Contributions

We provide formal definitions for subversion attacks against generic cryptographic primitives whose syntax allows for both the sending and receiving party to be subverted. Previous work in this area considered only subversion of the sender; our main contribution is to show that this assumption misses an important class of attack that targets the receiver. In this work, we describe how such an ASA against the receiver can be used to exfiltrate the (receiver's) key, which represents the most devastating attack from the point of view of an attacker. We show that this class of ASA can be applied to symmetric settings (authenticated encryption, message authentication codes, and data encapsulation mechanisms), as well as asymmetric settings (public key encryption and key encapsulation mechanisms). Our work brings together previous work targeting AEAD schemes [5] and MAC schemes [4] in a common framework, expanded to incorporate public key encryption.

Concretely, we alter the behaviour of the receiver's algorithm to leak information through (artificially induced) decryption error events – the subverted algorithm either rejects (particular, "trigger") valid ciphertexts or accepts (particular, "trigger") bogus ciphertexts. An adversary observing the receiver who is able to determine whether a ciphertext has been accepted or rejected learns some information; this subliminal channel can be used to exfiltrate the user's key. The assumption that a surveillance adversary is able to observe whether a receiver's algorithm implementation accepts or rejects a ciphertext is a mild one in many practical scenarios; for example, a decryption error may result in a network packet being dropped and automatically retransmitted.[1] A subverted algorithm could, furthermore, go beyond this by e.g. influencing timing information in future messages sent to the network. We conclude that this attack represents an attractive and easy to implement opportunity for a mass surveillance adversary.

1.1.1 AEAD

We first examine authenticated encryption with associated data (AEAD), a symmetric cryptographic primitive that offers the combined properties of confidentiality and message integrity. We show that our class of ASA applies to the decryption component of AEAD schemes, leaking the symmetric key. Our results stand in opposition to previous work [14,29,11] which proposed subversion resilience of a large class of AEAD schemes to which many if not all real-world constructions such as GCM, CCM and OCB belong, as long as their nonces are generated deterministically via a shared state maintained by both encryptor and decryptor. The crucial observation to resolve this apparent contradiction is that previous work has assumed, besides explicitly spelled out requirements like uniqueness of ciphertexts and perfect decryptability, implicit notions such as integrity of ciphertexts. In the ASA setting for AEAD where undermining the confidentiality of a scheme is the primary goal of an adversary, it seems just as natural to assume that the adversary is also willing to compromise the integrity guarantees as well.

1.1.2 MACs

We next show that our results apply equally in the setting of message authentication schemes (MACs). MACs provide a message authentication code or *tag* for a given message; conversely, given a message and a tag, the MAC provides verification that the tag was generated from the message (that is, that the tag is genuine). The security of a MAC is determined by the difficulty of forging tags. If no adversary can forge a tag, then a message with a correct tag

[1] Recent work on so-called partitioning oracles [47,2,3] relies on the ability to observe whether or not decryption succeeds and demonstrates that this is a realistic assumption in practice; for example, in the context of proxy servers a "logical side-channel" is observable as a port is opened when a ciphertext is accepted (and otherwise not).

must have been generated by the sender. An ASA against a MAC replaces either the tagging function (the generation of message authentication codes) or the verification function (checking that tags have been honestly generated) in such a way as to leak information to an adversary. Applying our attack to a MAC leaks the secret key to an adversary, allowing them to forge any tag. This is an attractive goal for an adversary in real world settings, as once integrity has been compromised this can often be leveraged to perform any number of other attacks, for example: enabling attacks against ("encrypt-then-MAC") confidentiality; getting users to accept compromised (authenticated) software updates; injecting malicious packets into (secured) communication streams to de-anonymise users.

1.1.3 PKE

Lastly, we show that public key encryption (PKE) is also vulnerable to our class of ASA. A public key (or asymmetric) encryption scheme allows secure communication between parties that have not shared a secret key with one another. PKE works by having two keys: the public key is used to encrypt messages and the private key is used to decrypt. The security of a PKE scheme is determined by the difficulty of determining any information about underlying messages for a given ciphertext.

Our ASA attacks on PKE require a fairly large number of ciphertexts to be sent and observed to reject erroneously in order for the private key to be exfiltrated. In practice, this condition will be met: consider a server that hosts traffic for a large number of clients. The server will have a private/public key pair which is held static over long periods of time. Observing the server receive ciphertexts from many clients will allow an adversary to witness a large enough amount of traffic to recover the server's private key, rendering all communications between clients and server compromised.

Due to the high overheads associated with PKE, symmetric encryption is better suited to bulk communication. In most practical settings, PKE is used to establish a shared secret between the sender and receiver, so that the shared secret may be used as a key for communicating via symmetric encryption. This notion of sending keys for symmetric encryption via public key methods is formalised as a key encapsulation mechanism (KEM). We show how our notions of subversion apply also to KEMs in Appendix A.1. KEMs are typically used together with a data encapsulation mechanism (DEM) in a so-called hybrid encryption scheme to PKE-encrypt messages. We give the definition of a DEM in Appendix A.2 for completeness.

1.2 Structure of this Document

We first describe related work in Section 2, focussing on ASAs that target symmetric encryption, PKE and MACs. Section 3 describes the notation used in this article. We give an abstract description of an ASA targeting generic cryptographic schemes consisting of a sender and receiver in Section 4, together with notions of undetectability (Section 4.1) and key recovery (Section 4.2). We also discuss hybrid subversion (Section 4.3), the idea that multiple algorithms (e.g., key generation and encryption) are subverted in tandem. In Section 5 we discuss authenticated encryption with associated data, giving syntax and security definitions: privacy in Section 5.1 and integrity in Section 5.2. We show that our notion of ASAs apply to AEAD schemes in Section 5.3. Section 6 discusses MACs, including the definition of integrity; Section 6.1 shows that our notion of ASAs apply to MAC schemes. Section 7 discusses PKE, giving syntax and security definitions; Section 7.1 shows that our notion of ASAs apply to PKE schemes. We describe our concrete subversion attack, targeting the receiver algorithm, in Section 8, together with an analysis of the undetectability and key recovery properties of our attack. We give two versions, a passive attack in Section 8.2 and an active attack in Section 8.3.

2 Related Work

2.1 Symmetric Encryption

BPR [14] demonstrate an attack against certain randomised encryption schemes that relies on influencing the randomness consumed in the course of encryption. Their attack, which they call the "biased-ciphertext attack", is a generic method that relies on rejection sampling. Randomness is resampled until ciphertexts satisfy a particular format (for example, implanting information in the least significant bits), resulting in a subliminal channel.

There is a tension for "Big Brother" between mounting a successful attack and being detected; clearly an attack that simply replaces the encryption algorithm with one that outputs the messages in plaintext would be devastating yet trivially detectable. BPR stipulate that ciphertexts generated with a subverted encryption algorithm should at the very least decrypt correctly with the unmodified decryption routine, in order to have some measure of resistance to detection. Furthermore, BPR define the success probability of a mass surveillance adversary in carrying out a successful attack, as well as the advantage of a user in detecting that a surveillance attack is taking place. The attack of BPR was later generalised by Bellare, Jaeger and Kane (BJK) [11] whose attack applies to all randomised schemes. Furthermore, whereas the attack of BPR is stateful and so vulnera-

ble to detection through state reset, the BJK attack is stateless. BJK [11] later formalised the goal of key recovery as the desired outcome of an ASA from the point of view of a mass surveillance adversary. Lastly, BPR also establish a positive result that shows that under certain assumptions, it is possible for authenticated encryption schemes to provide resistance against subversion attacks.

Degabriele, Farshim and Poettering (DFP) [29] critiqued the definitions and underlying assumptions of BPR. Their main insight is that the perfect decryptability —a condition mandated by BPR— is a very strong requirement and artificially limits the adversary's set of available strategies. In practice, a subversion with negligible detection probability, say 2^{-128}, should be considered undetectable.[2] As DFP note, decryption failures may happen for reasons other than a subverted encryption algorithm, and if they occur sporadically may easily go unnoticed. Thus a subverted encryption scheme that exhibits decryption failure with a very low probability is a good candidate for a practical ASA that is hard to detect. DFP demonstrate how this can be achieved with an input-triggered subversion, where the trigger is some message input that is difficult to guess, making detection practically impossible. Our work complements the trigger message approach of DFP by limiting ciphertext integrity and establishing a covert channel through decryption error events.

2.2 PKE

Yung and Young (YY) in [62] examine subverting asymmetric protocols in so-called "SETUP" attacks. Their core idea is to encode some information within the public key that allows the private key to be reconstructed. As a simple example, let the public key encode the encryption of the user's private key under the adversary's key. Subverted keys should be indistinguishable from real keys and only the adversary should be able to recover a user's private key from the subverted public key. As well as showing how to subvert RSA keys, YY also give examples of attacks against ElGamal, DSA and Kerberos. Later, Crépeau and Slakmon [27] gave an improved subversion attack against RSA which works by hiding half of the bits of p in the representation of the RSA modulus $N = pq$. Using Coppersmith's partial information attack [24], it is then possible to recover p and q.

For the prior work on symmetric encryption discussed above, the techniques can be translated naturally into a PKE setting. Attacks against the encryption algorithm of a PKE scheme however do not present an attractive attack to a mass surveillance adversary, as there is limited scope to undermine confidentiality. The covert channel usually has a bandwidth of a small number of bits per (subverted) ciphertext: not enough to leak the underlying messages. Leaking the private key would allow confidentiality to be broken completely, but the encryption algorithm does not have access to the private key. Chen, Huang and Yung [23] overcome these limitations by considering hybrid PKE constructions consisting of a KEM to send encapsulated session keys which are used for symmetric encryption with a DEM. Their non-generic attack applies to a particular class of practical KEM constructions and leaks session keys, that in turn break the security of the DEM. In contrast, for a PKE primitive not consisting of a hybrid KEM/DEM construction, targeting the decryption algorithm remains the only way to subvert the encryption/ decryption facility of a PKE scheme.

2.3 MACs

The only prior work on MAC subversion that we are aware of is by Al Mansoori, Baek, and Salah [1] who explore how a MAC component in the EAP-PSK wireless protocol could be subverted. After first arguing [1, §II.D] that randomised MAC schemes offer better protection against a kind of birthday attack, they restrict attention to precisely one corresponding construction (two-key CBC-MAC with a random translation of the second key, a scheme that already turned out to be broken in [45]) and show that the rejection-sampling based key-extraction techniques from [14] are applicable in this setting as well. We emphasise that our results reach far beyond this: our subversion attacks are generic (rather than being focused on one specific MAC) and we don't require exotic technical conditions like randomised tag generation.[3]

2.4 Further Work

Cryptographic reverse firewalls [49,34,48,61,21] represent an architecture to counter ASAs against asymmetric cryptography via trusted code in network perimeter filters. At a high level, the approach is for a trusted third party to re-randomise ciphertexts before transmission over a public network to destroy any subliminal messages. Fischlin and Mazaheri show how to construct ASA-resistant (asymmetric) encryption and signature algorithms given initial access to a trusted base scheme [38]. Their approach uses trusted samples to essentially perform re-randomisation of ciphertexts.

In a series of work, Russell, Tang, Yung and Zhou [55, 56,57,58] study ASAs on one-way functions, trapdoor one-way functions and key generation as well as defending randomised algorithms against ASAs using so-called watchdogs. The watchdog model allows a trusted party to test the implementation of a primitive for subversion, in a variety of

[2] This is analogous to the fundamental notion in cryptography that a symmetric encryption scheme be considered secure even in the presence of adversaries with negligible advantage.

[3] We are not aware of *any* randomised MAC of practical relevance.

different assumptions (e.g. on- or offline, black- or white-box access). Combiners are often used to provide subversion resilience, particularly in the watchdog model. A combiner [39,51] essentially combines the output from different algorithms (or runs of the same algorithm) in such a way as to produce secure (in this case, unsubverted) combined output as long as any one of the underlying outputs is secure. Aviram et al. [7] consider combining (potentially maliciously chosen) keys for Post-Quantum protocols such as TLS. Bemman, Chen and Jager [16] show how to construct a subversion-resilient KEM, using a variant of a combiner and a subversion resilient randomness generator. Their construction considers Russell et al.'s watchdog from a practical perspective, meaning an offline watchdog that runs in linear time. Another line of work, [37,9,32], examined backdoored hash functions, showing how to immunise hash functions against subversion.

Bellare, Kane and Rogaway [12] explore how large keys can prevent key exfiltration in the symmetric encryption setting. Bellare and Hoang [10] give PKE schemes that defend against the subversion of random number generators. The use of state reset to detect ASAs is studied by Hodges and Stebila [42]. Berndt and Liśkiewicz [17] reunite the fields of cryptography and steganography. Goh, Boneh, Pinkas and Golle [40] show how to add key recovery to the SSH and SSL/TLS protocols. Ateniese, Magri and Venturi [6] study ASAs on signature schemes. Berndt et al. consider ASAs against protocols such as TLS, WireGuard and Signal [18]. Dodis, Ganesh, Golovnev, Juels and Ristenpart [33] provide a formal treatment of backdooring PRGs, another form of subversion. This work was extended by Degabriele, Paterson, Schuldt and Woodage [30] to look at robust PRNGs with input. Camenisch, Drijvers and Lehmann [22] consider Direct Anonymous Attestation in the presence of a subverted Trusted Platform Module.

2.5 Cryptographic vs. Non-Cryptographic Subversion

In the literature on cryptography, the notion of an ASA assumes the malicious replacement of one or more algorithms of a scheme by a backdoored version, with the goal to leak key material, or at least to weaken some crucial security property. Different types of substitution attack appear in other areas of computing and communication. We discuss some examples in the following.

Program code in the domain of computer malware routinely modifies system functions to achieve its goals, where the latter comprises delivering some damaging payload, ensuring non-detection and thus survival of the malware on the host system, and in some cases even self-reproduction. Numerous techniques towards suitably modifying a host system have been developed and reported on by academic researchers and hackers. Standard examples include redirecting interrupt handlers, changing the program entry point of an executable file, and interfering with the OS kernel by overwriting its data structures [41].

Malicious modifications of implemented functionality are also a recognised threat in the hardware world. It is widely understood that circuit designers who do not possess the technical means to produce their own chips but instead outsource the production process to external foundries, risk that the chips produced might actually implement a maliciously modified version of what is expected. A vast number of independent options are known for when (within the production cycle) and how (functionally) subversions could be conducted. For instance, the survey provided in [19] reports that circuit design software (CAD) could be maliciously altered, that foundries could modify circuits before production, and that after production commercial suppliers could replace legitimate chips by modified ones. Further, [19] suggests that appealing types of functionality modification include deviating from specification when particular input trigger events are recognised, and/or to leak values of vital internal registers via explicitly implemented side channels. Any such technique (or combination thereof) has an individual profile regarding the associated costs and attack detectability. Which of the many options is most preferable depends on the specific attack scenario and target.

We refer to the software and hardware based subversion techniques discussed above as "technology driven". This is in contrast to the techniques considered in this paper which we refer to as "semantics driven". We consider the two approaches orthogonal: Our (semantics driven) proposed subversion *can* be implemented using techniques from e.g. [41, 19] (but likewise also through standard methods), and technology driven subversion proposals *can* be applied against cryptographic implementations (but likewise also against any other interesting target functionality). Our semantics driven approach in fact aims to maximise technology independence. As a consequence, the line of attacks proposed in this paper can be implemented easily in software (e.g. in libraries or drop-in code), in hardware (e.g. in ASICs and FPGAs), and in mixed forms (e.g. firmware-programmed microcontrollers). The strategy to achieve this independence is to base the attacks and corresponding notions of (in)security on nothing but the abstract functionalities of the attacked scheme as they are determined by their definitions of syntax and correctness.

As the technology driven and semantics driven approaches are independent, they can in particular be combined. This promises particularly powerful subversions. For instance, consider that virtually all laptops and desktop PCs produced in the past decade are required to have an embedded trusted platform module (TPM) chip that supports software components (typically boot loaders and operating systems) with *trusted* cryptographic services. In detail, software can in-

teract with a TPM chip through standardised API function calls and have cryptographic operations applied to provided inputs, with all key material being generated and held exclusively in the TPM. As TPMs are manufactured in hardware, it seems that the (technology driven) subversion options proposed in [19] would be particularly suitable. However, as most of the attacks from [19] require physical presence of the adversary (e.g., to provide input triggers via specific supply voltage jitters or for extracting side channel information by operating physical probes in proximity of the attacked chip), only those options seem feasible where all attack conditions and events can be controlled and measured via the software interface provided by the API. This is precisely what our semantics driven attacks provide. We thus conclude by observing that dedicated cryptographic hardware like TPMs can only be trusted if *extreme* care is taken during design and production. While our article lays open the most general and clean line of attack, other attacks might exist as well.

2.5.1 Discussion

As the discussion of cryptographic ("semantics driven") vs. non-cryptographic ("technology driven") subversion shows, achieving security against adversaries mounting ASAs is difficult, and essentially reduces to assuming trust in particular components or architectures. The three main theoretical approaches to preventing or mitigating against ASAs in the literature, discussed above in Section 2.4, are reverse firewalls, self-guarding protocols and watchdogs. We note that these approaches apply in the main to asymmetric primitives, and so (appropriately adapted to target receiver algorithms) would be suitable to defend against our attack against asymmetric schemes in Section 7.

Defending against our attacks on AEAD and MACs is more difficult. We note that the watchdog model applies in theory, while reverse firewalls and self-guarding approaches are ineffective against symmetric primitives. The watchdog model considers splitting a primitive into constituent algorithms that are run as subroutines by a trusted "amalgamation" layer. This allows the constituent algorithms to be individually checked and sanitised. Considering the verification algorithm of a MAC scheme as an example, the canonical approach of recalculating and checking the tag is modelled by letting the verification algorithm be a trusted amalgamation of the tagging algorithm with an identity test. The tagging algorithm typically runs a hash function as a subroutine, and so applying results from [37,9,32] would allow for the claim that the verification algorithm can be made subversion-resilient in the watchdog model. The assumption of a trusted amalgamation is precisely what makes our attack infeasible, but this assumption is questionable in real world settings. In particular, as we discussed above, the presence of non-cryptographic vectors makes this assumption unlikely to hold in practice.

Lastly, we note that none of the theoretical approaches are fully satisfying, requiring strong or impractical assumptions. Indeed, it is telling that there are no implementations of subversion-resilient primitives to date, although some recent work seems promising in this regard [21,16]. The best defense seems to be the unglamorous task of minimising risk by implementing a variety of control mechanisms across the whole infrastructure, in a process of security management. In particular: software implementations could be protected by measures including regular integrity tests and secure boot, hardware implementations could be protected by technical controls such as threshold implementations or testing amplification [36], and both cases can be strengthened by relying on open source implementations and verified supply chains. Whilst such measures can go some way towards minimising risk, we emphasise that there are no security guarantees.

3 Notation

We refer to an element $x \in \{0,1\}^*$ as a string, and denote its length by $|x|$. The set of strings of length l is denoted $\{0,1\}^l$. By ε we denote the empty string. For $x \in \{0,1\}^*$ we let $x[i]$ denote the i-th bit of x, with the convention that we count from 0, i.e., we have $x = x[0]\ldots x[|x|-1]$. We use Iverson brackets $[\cdot]$ to derive bit values from Boolean conditions: For a condition C we have $[C] = 1$ if C holds; otherwise we have $[C] = 0$.

We use code-based notation for probability and security experiments. We write $\leftarrow$ for the assignment operator (that assigns a right-hand-side value to a left-hand-side variable). If S, S' are sets, we write $S \xleftarrow{\cup} S'$ shorthand for $S \leftarrow S \cup S'$. If S is a finite set, then $s \leftarrow_\$ S$ denotes choosing s uniformly at random from S. For a randomised algorithm A we write $y \leftarrow_\$ \mathsf{A}(x_1, x_2, \ldots)$ to denote the operation of running A with inputs $x_1, x_2, \ldots$ and assigning the output to variable y. We denote a γ-biased Bernoulli trial by $B(\gamma)$, i.e., a random experiment with possible outcomes 0 or 1 such that $\Pr[b \leftarrow_\$ B(\gamma) : b = 1] = \gamma$. The assignments $b \leftarrow_\$ \{0,1\}$ and $b \leftarrow_\$ B(1/2)$ are thus equivalent. We use superscript notation to indicate when an algorithm (typically an adversary) is given access to specific oracles. An experiment terminates with a "stop with x" instruction, where value x is understood as the outcome of the experiment. We write "win" ("lose") as shorthand for "stop with 1" ("stop with 0"). We write "require C", for a Boolean condition C, shorthand for "if not C: lose". (We use require clauses typically to abort a game when the adversary performs some disallowed action, e.g. one that would lead to a trivial win.) The ":=" operator creates a symbolic definition; for instance, the code line

"$A := E$" does *not* assign the value of expression E to variable A but instead introduces symbol A as a new (in most cases abbreviating) name for E.

4 Notions of Subversion Attacks

We consider subversions of the algorithms of cryptographic schemes. Abstractly, we consider a cryptographic scheme $\Pi = (\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R})$ consisting of three components: a key generation algorithm together with an algorithm on the sender side and an algorithm on the receiver side. Where the cryptographic scheme is an encryption scheme, $\Pi.\mathsf{S}$ represents encryption and $\Pi.\mathsf{R}$ decryption; when we consider message authentication schemes, the corresponding components represent tagging and verification.

We give a generic syntax to the scheme Π as follows: key generation $\Pi.\mathsf{gen}$ outputs a key pair $(k_\mathsf{S}, k_\mathsf{R}) \in \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}$; the sender algorithm has associated input and output spaces $\mathcal{X}, \mathcal{Y}$ and takes as input a key $k_\mathsf{S} \in \mathcal{K}_\mathsf{S}$ and $x \in \mathcal{X}$, outputting $y \in \mathcal{Y}$; the receiver algorithm has associated input and output spaces $\mathcal{Y}, \mathcal{X}'$ (respectively). We note that $\mathcal{X} \subsetneq \mathcal{X}'$; in particular, $\perp \in \mathcal{X}' \setminus \mathcal{X}$. The receiver algorithm takes as input a key $k_\mathsf{R} \in \mathcal{K}_\mathsf{R}$ and $y \in \mathcal{Y}$, outputting $x \in \mathcal{X}'$; the special symbol $\perp$ is used to indicate failure. A shortcut notation for this syntax is

$$\Pi.\mathsf{gen} \to \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}, \quad \mathcal{K}_\mathsf{S} \times \mathcal{X} \to \Pi.\mathsf{S} \to \mathcal{Y}, \quad \text{and} \quad \mathcal{K}_\mathsf{R} \times \mathcal{Y} \to \Pi.\mathsf{R} \to \mathcal{X}'.$$

A scheme Π is said to be δ-correct if for all $(k_\mathsf{S}, k_\mathsf{R}) \leftarrow \Pi.\mathsf{gen}$ and $x \in \mathcal{X}$ and $y \leftarrow \Pi.\mathsf{S}(k_\mathsf{S}, x)$ and $x' \leftarrow \Pi.\mathsf{R}(k_\mathsf{R}, y)$ we have

$$\Pr\left[x' \neq x\right] \leq \delta\,,$$

where the probability is over all random coins involved. In the case that $\delta = 0$, the scheme is said to be perfectly correct. We note that this generic syntax applies to symmetric encryption (Section 5) and message authentication (Section 6), as well as to public key encryption (Section 7), where for the symmetric case we require $k_\mathsf{S} = k_\mathsf{R}$.

In the following, we give formal definitions for subversion of key generation, sender and receiver algorithms, together with the notion of *undetectability* (UD). In a nutshell, a subversion is undetectable if distinguishers with black-box access to either the original scheme or to its subverted variant cannot tell the two apart. A subversion should exhibit a dedicated functionality for the subverting party, but simultaneously be undetectable for all others. This apparent contradiction is resolved by parameterising the subverted algorithm with a secret subversion key, knowledge of which enables the extra functionality. (The same technique is used in most prior work, starting with [14].) In what follows we denote the corresponding subversion key spaces with $\mathcal{I}_{\mathsf{gen}}, \mathcal{I}_\mathsf{S}$ and $\mathcal{I}_\mathsf{R}$.

In this section we also specify, by introducing notions of key recoverability, how we measure the quality of a subversion from the point of view of the subverting adversary (who is assumed to know the subversion keys).

4.1 Undetectable Subversion

We first define undetectability notions for subverted key generation, sender and receiver algorithms separately. We then offer a joint definition.

SUBVERTED KEY GENERATION. A subversion of the key generation algorithm $\Pi.\mathsf{gen}$ of a cryptographic scheme consists of a finite index space $\mathcal{I}_{\mathsf{gen}}$ and a family of algorithms $\mathcal{G}en = \{\Pi.\mathsf{gen}_i\}_{i \in \mathcal{I}_{\mathsf{gen}}}$ with

$$\Pi.\mathsf{gen}_i \to \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}.$$

That is, for all $i \in \mathcal{I}_{\mathsf{gen}}$ the algorithm $\Pi.\mathsf{gen}_i$ can syntactically replace the algorithm $\Pi.\mathsf{gen}$.

As a security property we require that also the observable behaviour of $\Pi.\mathsf{gen}$ and $\Pi.\mathsf{gen}_i$ be effectively identical (for uniformly chosen $i \in \mathcal{I}_{\mathsf{gen}}$). This is formalised via the games $\mathrm{UDG}^0, \mathrm{UDG}^1$ in Fig. 1 (left). For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathrm{udg}}_\Pi(\mathcal{A}) := |\Pr[\mathrm{UDG}^1(\mathcal{A})] - \Pr[\mathrm{UDG}^0(\mathcal{A})]|$$

and say that family $\mathcal{G}en$ undetectably subverts algorithm $\Pi.\mathsf{gen}$ if $\mathsf{Adv}^{\mathrm{udg}}_\Pi(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

SUBVERTED SENDER. A subversion of the sender algorithm $\Pi.\mathsf{S}$ of a cryptographic scheme consists of a finite index space $\mathcal{I}_\mathsf{S}$ and a family $\mathcal{S} = \{\mathsf{S}_i\}_{i \in \mathcal{I}_\mathsf{S}}$ of algorithms

$$\mathcal{K}_\mathsf{S} \times \mathcal{X} \to \Pi.\mathsf{S}_i \to \mathcal{Y}.$$

That is, for all $i \in \mathcal{I}_\mathsf{S}$ the algorithm $\Pi.\mathsf{S}_i$ can syntactically replace the algorithm $\Pi.\mathsf{S}$.

As a security property we also require that the observable behaviour of $\Pi.\mathsf{S}$ and $\Pi.\mathsf{S}_i$ be effectively identical (for uniformly chosen $i \in \mathcal{I}_\mathsf{S}$). This is formalised via the games $\mathrm{UDS}^0, \mathrm{UDS}^1$ in Fig. 1 (centre). Note that, in contrast to prior work like [14,29], our distinguishers are given free choice over the keys to be used.[4] For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathrm{uds}}_\Pi(\mathcal{A}) := |\Pr[\mathrm{UDS}^1(\mathcal{A})] - \Pr[\mathrm{UDS}^0(\mathcal{A})]|$$

and say that family $\mathcal{S}$ undetectably subverts algorithm $\Pi.\mathsf{S}$ if $\mathsf{Adv}^{\mathrm{uds}}_{\mathsf{PKE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

[4] In [14,29], undetectability is defined with respect to uniform keys. As code auditors and other security researchers looking for subversion attacks can specify keys during black-box testing according to their preferred distribution, we consider uniform-key constraints a rather severe limitation of undetectability notions.

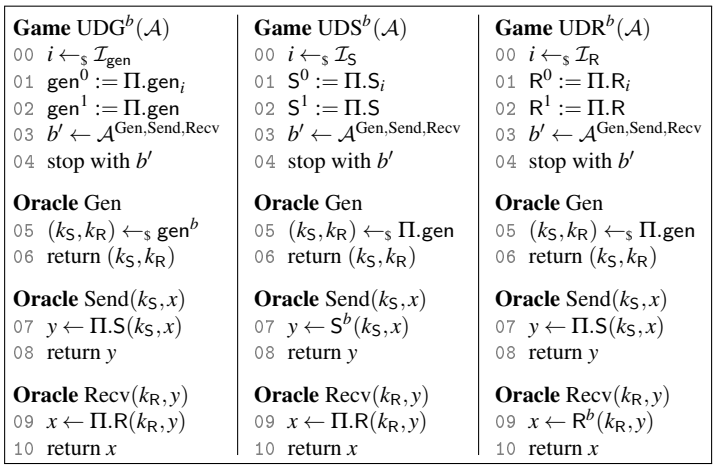

Fig. 1 Games UDG, UDS and UDR modelling undetectability for the subversion of (respectively) key generation, sender and receiver algorithms for a cryptographic scheme Π. See Section 3 for the meaning of ":=". Note that in each game, the two unsubverted oracles are actually redundant.

SUBVERTED RECEIVER. A subversion of the receiver algorithm $\Pi.\mathsf{R}$ of a cryptographic scheme consists of a finite index space $\mathcal{I}_\mathsf{R}$ and a family $\mathcal{R} = \{\Pi.\mathsf{R}_i\}_{i\in\mathcal{I}_\mathsf{R}}$ of algorithms

$$\mathcal{K}_\mathsf{R} \times \mathcal{Y} \to \Pi.\mathsf{R}_i \to \mathcal{X}'.$$

That is, for all $i \in \mathcal{I}_\mathsf{R}$ the algorithm $\Pi.\mathsf{R}_i$ can syntactically replace the algorithm $\Pi.\mathsf{R}$.

As a security property we also require that the observable behaviour of $\Pi.\mathsf{R}$ and $\Pi.\mathsf{R}_i$ be effectively identical (for uniformly chosen $i \in \mathcal{I}_\mathsf{R}$). This is formalised via the games $\mathrm{UDR}^0, \mathrm{UDR}^1$ in Fig. 1 (right). For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathrm{udr}}_\Pi(\mathcal{A}) := |\Pr[\mathrm{UDR}^1(\mathcal{A})] - \Pr[\mathrm{UDR}^0(\mathcal{A})]|$$

and say that family $\mathcal{R}$ undetectably subverts algorithm $\Pi.\mathsf{R}$ if $\mathsf{Adv}^{\mathrm{udr}}_{\mathsf{PKE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

The above undetectability notions demand that subversions do not change the observable behaviour of the key generation, sender and receiver algorithms. A consequence of this is that none of the correctness or security properties of the scheme are noticeably harmed by subversion.

4.1.1 Hybrid Subversion of Key Generation, Sender and Receiver algorithms

We give a joint definition of undetectability, in the case where the key generation, sender and receiver algorithms are subverted. This is the most general definition; in particular contexts it may not be appropriate to consider subversion of a particular algorithm – we discuss this below in Sections 4.3, 5.3, 6.1.1 and 7.1.1.

Game UD^b in Fig. 2 (left) combines games UDG^b, UDS^b and UDR^b into one. We define

$$\mathsf{Adv}^{\mathrm{ud}}_\Pi(\mathcal{A}) := |\Pr[\mathrm{UD}^1(\mathcal{A})] - \Pr[\mathrm{UD}^0(\mathcal{A})]|.$$

By a hybrid argument, for all adversaries $\mathcal{A}$ there exist adversaries $\mathcal{A}', \mathcal{A}'', \mathcal{A}'''$ such that

$$\mathsf{Adv}^{\mathrm{ud}}_\Pi(\mathcal{A}) \le \mathsf{Adv}^{\mathrm{udg}}_\Pi(\mathcal{A}') + \mathsf{Adv}^{\mathrm{uds}}_\Pi(\mathcal{A}'') + \mathsf{Adv}^{\mathrm{udr}}_\Pi(\mathcal{A}''').$$

4.2 Subversion Leading to Key Recovery

We observed above that if any of the components $\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R}$ of a cryptographic scheme Π is undetectably subverted, with uniformly chosen indices $i_\mathsf{gen}, i_\mathsf{S}, i_\mathsf{R}$ that remain unknown to the participants, then all security guarantees are preserved from the original scheme. This may be different if (any of) $i_\mathsf{gen}, i_\mathsf{S}, i_\mathsf{R}$ are known to an attacking party, and indeed we assume that mass-surveillance attackers leverage such knowledge to conduct attacks. For any cryptographic scheme, the most devastating attack goal for an attacker is key recovery (KR): Users generate keys using their key generation algorithm $(k_\mathsf{S}, k_\mathsf{R}) \leftarrow_\$ \Pi.\mathsf{gen}_{i_\mathsf{gen}}$.[5] Generated secret keys are kept hidden, and the adversary aims at recovering these keys through the subversion. Note that in the symmetric case, $k_\mathsf{S} = k_\mathsf{R}$, whereas in the asymmetric case the receiver's key k_R represents the private key. In either case, the value k_R is the target of a KR adversary.

We formalise this attack goal in two versions. The KRP game in Fig. 2 (centre) assumes a passive attack in which the adversary cannot manipulate inputs or outputs (typically representing messages or ciphertexts) to the sender or receiver, and the KRA game in Fig. 2 (right) assumes an active attack in which the adversary can inject and test arbitrary receiver inputs (which potentially correspond to sender outputs). In both cases, with the aim of closely modelling real-world settings, we restrict the adversary's influence on the

[5] To preserve generality, our syntax suggests that key generation is subverted, however this need not be the case. Simply set $\Pi.\mathsf{gen}_{i_\mathsf{gen}} := \Pi.\mathsf{gen}$ for all $i_\mathsf{gen} \in \mathcal{I}_\mathsf{gen}$. This applies similarly to $\Pi.\mathsf{S}$ and $\Pi.\mathsf{R}$.

sender inputs x by assuming a stateful "message sampler" algorithm MS (reflecting the fact that, in the contexts we consider, inputs to $\Pi.\mathsf{S}$ typically represent messages) that produces the inputs to $\Pi.\mathsf{S}$ used throughout the game. The syntax of this message sampler is

$$\Sigma \times A \to \mathrm{MS} \to \Sigma \times \mathcal{X} \times B, \\ (\sigma, \alpha) \mapsto \mathrm{MS}(\sigma, \alpha) = (\sigma', x, \beta),$$

where $\sigma, \sigma' \in \Sigma$ are old and updated state, input $\alpha \in A$ models the influence that the adversary may have on message generation, and output $\beta \in B$ models side-channel outputs. In Fig. 2 we write $\diamond$ for the initial state. Note that while we formalise the inputs α and the outputs β for generality (so that our models cover most real-world applications), our subversion attacks are independent of them.[6] For any message sampler MS and adversary $\mathcal{A}$ we define the advantages

$$\mathsf{Adv}^{\mathrm{krp}}_{\Pi,\mathrm{MS}}(\mathcal{A}) := \Pr[\mathrm{KRP}(\mathcal{A})] \\ \text{and } \mathsf{Adv}^{\mathrm{kra}}_{\Pi,\mathrm{MS}}(\mathcal{A}) := \Pr[\mathrm{KRA}(\mathcal{A})].$$

We say that subversion family $\mathcal{G}en, \mathcal{S}, \mathcal{R}$ is key recovering for passive attackers if for all practical MS there exists a realistic adversary $\mathcal{A}$ such that $\mathsf{Adv}^{\mathrm{krp}}_{\mathrm{MS}(\mathcal{A})}$ reaches a considerable value (e.g., 0.1).[7] The key recovery notion for active attackers is analogous.

4.2.1 Discussion

We note that the adversary need not necessarily exfiltrate each individual bit of the user's key,[8] in order to successfully recover it – this is implicit in our definitions of key recovery. To formalise this, we let the "leakage key" $k_\ell \in \{0,1\}^\lambda$ be a string such that knowledge of k_ℓ is sufficient for an adversary to break the security of the primitive. At worst, from the perspective of the adversary, the leakage key may simply be the bit representation of the user's key. We note that in practice a leakage key consisting of *most* of the user's key is sufficient for an adversary to recover the full key using brute force; the exact number of bits to be brute forced would depend on the context and would involve a trade-off for the adversary. Nevertheless, the notion is intuitively clear.

Furthermore, in some contexts there may be some redundancy or structure that allows for a shorter leakage key. As an example, one may consider DES keys as being 64-bit strings with 8 bits of redundancy, so that an effective leakage key would be of size 56 bits. As another example, the private key in RSA encryption is knowledge of the factorisation of the public modulus $N = pq$. Supposing that the modulus N can be represented using $n = \lfloor \log N \rfloor$-bits, one may consider RSA private keys as being $n/2 = \lfloor \log p \rfloor$-bit strings. However, knowledge of around half the bits of p is sufficient to be able to factorise N using Coppersmith's partial information attack [24], so that an effective leakage key might have length $\lambda = \lfloor \log p \rfloor / 2$.[9]

A different approach might be to leak, for example, the seed of a pseudo-random number generator. We discuss breaking security without extracting the full key further in Section 4.4.

4.3 Hybrid Subversion

Previous work on subversion has looked at either subverted key generation[10] or subverted encryption/ decryption, but not considered the case where these are subverted in tandem. For key generation, this has meant that the subverted algorithm needs to leak the whole key in a single operation. This setting was studied by Young and Yung [62] under the name "kleptography", and they showed how it is possible to subvert key generation such that the adversary is able to recover the private key sk from the public key pk (together with any public parameters and knowledge of secret trapdoor information). They show how such attacks against key generation could look in the case of RSA and ElGamal cryptosystems. Such subversion imposes a large cost on the subverter: requiring that all key bits are leakable in one operation means that the subverted keys are given some structure (e.g. the public key is the encryption of the secret key under the attacker's key). This imposes significant overhead, and would likely lead to detection in a real world setting (using either timing information, code review or hardware inspection). Considering the subversion of key generation and sender/receiver algorithms in tandem, it is possible to reduce this overhead.

Generically, this tandem subversion can be achieved by subverting key generation to produce weaker keys and combining this with a subverted sender and/or receiver that provides a subliminal channel. Consider a subverted key generation algorithm $\Pi.\mathsf{gen}_i$ that outputs (receiver keys) in some reduced key set $\widetilde{\mathcal{K}}_\mathsf{R} \subset \mathcal{K}_\mathsf{R}$. The smaller this subverted key set $\widetilde{\mathcal{K}}_\mathsf{R}$, the less information needs to be leaked via the subliminal channel. One method to implement such a subverted key generation algorithm is to use the rejection sampling method described in Section 2.1, so that $\Pi.\mathsf{gen}_i$ runs the unsubverted

[6] ... meaning that the reader may safely choose to ignore them.

[7] Our informal notions ("realistic" and "practical") are easily reformulated in terms of probabilistic polynomial-time (PPT) algorithms for readers who prefer a treatment in the asymptotic framework. Given that asymptotic notions don't reflect practice particularly well, we prefer to use the informal terms.

[8] or a bit representation thereof, if it is not a bit string

[9] We note that in practice the security for an RSA modulus of size n is far less than $n/2$ bits; for example, an RSA modulus of size 1024 is believed to have security at most 80 bits [8], corresponding to the computational effort required to factorise an RSA modulus.

[10] and potentially, the associated public parameters if those form part of the formalisation used.

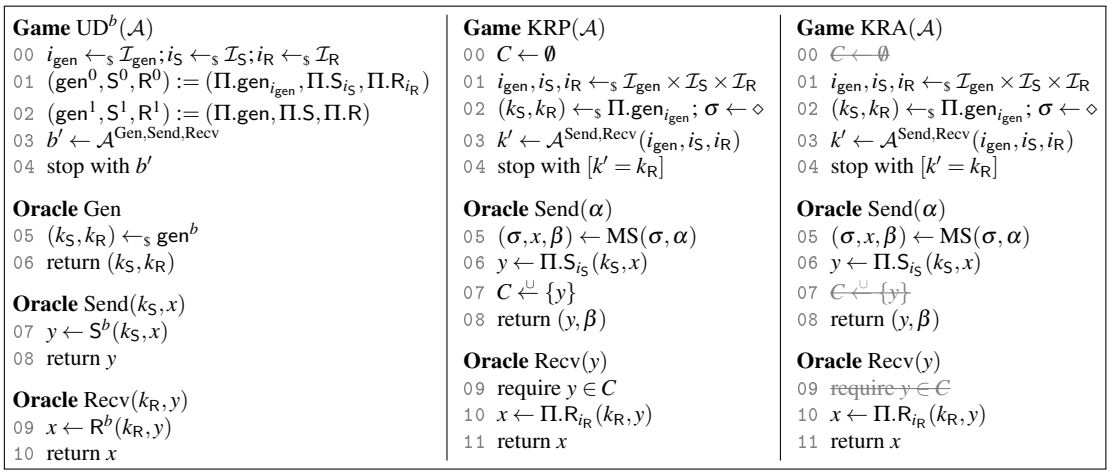

Fig. 2 Left: Game UD modelling hybrid subversion undetectability for a cryptographic scheme Π. Note that not all of the algorithms need necessarily be subverted, although the syntax allows for this. See the discussion at Sections 5.3, 6.1.1 and 7.1.1. **Centre, Right:** Games KRP and KRA modelling key recoverability for passive and active attackers, respectively. Note that the adversary's aim is to recover the receiver's key k_{R}, as in both symmetric and asymmetric settings this value is secret.

algorithm $\Pi.\mathsf{gen}$ as a subroutine and resamples until keys are in $\widetilde{\mathcal{K}}_{\mathsf{R}}$. There are certainly more targeted attacks that take into account the specific structure of keys being generated – and that may leverage more specific attacks than the generic weakening of keys.[11]

4.4 Breaking Security without Extracting the Full Key

The KRA and KRP notions introduced in Section 4.2 assume that key recovery is the ultimate goal in subversion. This suggests that longer keys make a scheme more resilient, an approach explored in big key cryptography [12]. In practice, it may be more efficient to exploit non-generic features of a particular scheme to minimise the information to be leaked. In this section, we will consider AEAD schemes as an illustrative example.

As we detail, many current AEAD schemes have inner building blocks that maintain their own secret values, and scaling up key sizes does not automatically also increase the sizes of these internal values. We note that proposed ASAs against AEAD schemes (including our attacks presented in Section 8) can easily be adapted to leak this internal information instead of the key. As the recovery of such values might not always directly lead to full message recovery, the assessment of whether the resulting overall attack is more or less effective than our generic attacks has to be made on a per scheme basis. We exemplify this on the basis of two of the currently best-performing AES-based AEAD schemes: GCM [35] and OCB3 [46]. In both cases, the size of the crucial internal value and the block size of the cipher have to coincide and the latter value is fixed to 128 bits for AES (independently of key size).

AES-GCM. We consider the following abstraction of GCM. The AEAD key k is used directly to create an instance E of the AES blockcipher. To encrypt a message m with respect to associated data d and nonce n, E is operated in counter mode, giving a pad $E(n+1) \,\|\, E(n+2) \,\|\, \ldots$, where a specific nonce encoding ensures there are no collisions between counter values of different encryption operations. The first part c_1 of the ciphertext $c = c_1 c_2$ is obtained by XORing the pad into the message, and finally the authentication tag c_2 is derived by computing $c_2 \leftarrow E(n) + H_h(d, c_1)$. Here H_h is an instance of a universal hash function H indexed (that is, keyed) with the 128-bit value $h = E(0^{128})$. Concretely, $H_h(d, c_1) = \sum_{i=1}^{l} v_i h^{l-i+1}$, where coefficients $v_1, \ldots, v_l$ are such that a prefix $v_1 \ldots v_j$ is a length-padded copy of the associated data d, the middle part $v_{j+1} \ldots v_{l-1}$ is a length-padded copy of ciphertext component c_1, and the last item v_l is an encoding of the lengths of d and c_1. The addition and multiplication operations deployed in this computation are those of a specific representation of the Galois field $\mathrm{GF}(2^{128})$.

In executing a practical ASA against AES-GCM, it might suffice to leak the value h (which has length 128-bits independently of the AES key length, and furthermore stays invariant across encryption operations). The insight is that if the key of a universal hash function is known, then it becomes trivial to compute collisions. Concretely, assume the adversary is provided with the AES-GCM encryption $c = c_1 c_2 = \mathsf{enc}(k, n, d, m)$ for unknown k, m but chosen d, n. Then by the above we have $c_2 = R + \sum_{i=1}^{j} v_i h^{l-i+1}$ where the coefficients $v_1 \ldots v_j$ are an encoding of d and R is some residue. If, having been successfully leaked by the ASA, the

[11] For instance, for the specific case of ElGamal encryption, [28] recently demonstrated that the key generation algorithms of relevant open-source implementations produced public keys that exhibit a number-theoretical structure that considerably reduces their effective key length.

internal value h is known, by solving a linear equation it is easy to find an associated data string $d' \neq d$, $|d'| = |d|$, such that for its encoding $v'_1 \dots v'_j$ we have $\sum_{i=1}^{j} v'_i h^{l-i+1} = \sum_{i=1}^{j} v_i h^{l-i+1}$. Overall this means that we have found $d' \neq d$ such that $\mathsf{enc}(k,n,d',m) = c = \mathsf{enc}(k,n,d,m)$. In a CCA attack the adversary can thus query for the decryption of c with associated data d' and nonce n, and thus fully recover the target message m. We finally note that this attack can be directly generalised to one where also the c_1 and c_2 components are modified, resulting in the decryption of a message $m' \neq m$ for which the XOR difference between m and m' is controlled by the adversary.

OCB3. Multiple quite different versions of the OCB encryption scheme exist [43], but a common property is that the associated data input is incorporated via "ciphertext translation" [54]. To encrypt a message m under key k with associated data d and nonce n, in a first step the message m is encrypted with a pure AE scheme[12]) to an intermediate ciphertext $c^* \leftarrow \mathsf{enc}^*(k,n,m)$. Then to obtain the final ciphertext c, a pseudo-random function value $F_k(d)$ of the associated data string is XORed into the trailing bits of c^*. Concretely, in OCB3 we have $F_k(d) = \sum_{i=1}^{l} E(v_i + C_i)$ where all addition operations are XOR combinations of 128 bit values, $E(\cdot)$ stands for AES enciphering with key k, values $v_1, \dots, v_l$ represent a length-padded copy of associated data d, and coefficients $C_1, \dots, C_l$ are (secret) constants deterministically derived from the value $L = E(0^{128})$.

In the context of an ASA we argue that it is sufficient to leak the 128 bit value L. The attack procedure is, roughly, as in the AES-GCM case. Assume the adversary is provided with the OCB3 encryption $c = \mathsf{enc}(k,n,d,m)$ for unknown k,m but chosen d,n, and assume the adversary knows L and thus $C_1, \dots, C_l$. Now let $1 \leq s < t \leq l$ be any two indices, let $\Delta = C_s + C_t$ and let $d' \neq d$, $|d'| = |d|$, be the associated data string with encoding $v'_1, \dots, v'_l$ such that we have $v'_s = v_t + \Delta$ and $v'_t = v_s + \Delta$ and $v'_i = v_i$ for all $i \neq s,t$. Then we have $E(v'_s + C_s) = E(v_t + \Delta + C_s) = E(v_t + C_t)$ and $E(v'_t + C_t) = E(v_s + \Delta + C_t) = E(v_s + C_s)$, which leads to $F_k(d) = F_k(d')$ and ultimately $\mathsf{enc}(k,n,d',m) = \mathsf{enc}(k,n,d,m)$. In a CCA attack environment, this can immediately be leveraged to the full recovery of m. As in the AES-GCM case, we note that many variants of our attack exist (against all versions of OCB), including some that manipulate message bits in a controlled way.

5 AEAD Schemes

We recall standard notions of (deterministic) nonce-based AEAD, as per [54], and study how to adapt them to the ASA setting. Formally, a scheme AEAD providing authenticated encryption with associated data consists of algorithms $\mathsf{AEAD.gen}, \mathsf{AEAD.enc}, \mathsf{AEAD.dec}$. The scheme has associated spaces $\mathcal{K}, \mathcal{N}, \mathcal{D}, \mathcal{M}, \mathcal{C}$. The key generation algorithm AEAD.gen outputs a key $k \in \mathcal{K}$. The encryption algorithm AEAD.enc takes key $k \in \mathcal{K}$, nonce $n \in \mathcal{N}$, associated data $d \in \mathcal{D}$ and message $m \in \mathcal{M}$, to produce ciphertext $c \in \mathcal{C}$. The decryption algorithm AEAD.dec takes key k, nonce $n \in \mathcal{N}$, associated data $d \in \mathcal{D}$ and ciphertext $c \in \mathcal{C}$ to output either a message $m \in \mathcal{M}$ or the special symbol $\perp \notin \mathcal{M}$ to indicate rejection. A shortcut notation for this syntax is

$$\mathsf{AEAD.gen} \to \mathcal{K}, \quad \mathcal{K} \times \mathcal{N} \times \mathcal{D} \times \mathcal{M} \to \mathsf{AEAD.enc} \to \mathcal{C}$$
$$\text{and} \quad \mathcal{K} \times \mathcal{N} \times \mathcal{D} \times \mathcal{C} \to \mathsf{AEAD.dec} \to \mathcal{M} \cup \{\perp\}.$$

Scheme AEAD is said to be δ-correct if for $k \leftarrow_{\$} \mathsf{AEAD.gen}$ and $c \leftarrow \mathsf{AEAD.enc}(k,n,d,m)$ for some (n,d,m) and $m' \leftarrow \mathsf{AEAD.dec}(k,n,d,c)$ the probability that $m' \neq m$ is upper-bounded by δ, where the probability is over all coins involved.

Game $\mathsf{IND\text{-}CCA}^b(\mathcal{A})$
00 $C \leftarrow \emptyset, N \leftarrow \emptyset$
01 $k \leftarrow_{\$} \mathsf{AEAD.gen}$
02 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
03 stop with b'

Oracle $\mathrm{Enc}(n,d,m^0,m^1)$
04 require $n \notin N$
05 $N \xleftarrow{\cup} \{n\}$
06 $c \leftarrow \mathsf{AEAD.enc}(k,n,d,m^b)$
07 $C \xleftarrow{\cup} \{(n,d,c)\}$
08 return c

Oracle $\mathrm{Dec}(n,d,c)$
09 require $(n,d,c) \notin C$
10 $m \leftarrow \mathsf{AEAD.dec}(k,n,d,c)$
11 return m

Game $\mathsf{subIND\text{-}CCA}^b(\mathcal{A})$
00 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
01 $C \leftarrow \emptyset, N \leftarrow \emptyset$
02 $k \leftarrow \mathsf{AEAD.gen}_{i_{\mathsf{gen}}}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
04 stop with b'

Oracle $\mathrm{Enc}(n,d,m^0,m^1)$
05 require $n \notin N$
06 $N \xleftarrow{\cup} \{n\}$
07 $c \leftarrow \mathsf{AEAD.enc}_{i_{\mathsf{S}}}(k,n,d,m^b)$
08 $C \xleftarrow{\cup} \{(n,d,c)\}$
09 return c

Oracle $\mathrm{Dec}(n,d,c)$
10 require $(n,d,c) \notin C$
11 $m \leftarrow \mathsf{AEAD.dec}_{i_{\mathsf{R}}}(k,n,d,c)$
12 return m

Fig. 3 Games modelling indistinguishability under chosen-ciphertext attacks (IND-CCA), and subverted indistinguishability under chosen-ciphertext attacks (subIND-CCA) for an authenticated encryption scheme with associated data AEAD.

5.1 IND-CCA

We formalise indistinguishability under chosen-ciphertext attack for an AEAD scheme via the game IND-CCA in Fig. 3 (left). For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{AEAD}}(\mathcal{A}) := \left|\Pr\left[\mathsf{IND\text{-}CCA}^0(\mathcal{A})\right] - \Pr\left[\mathsf{IND\text{-}CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme AEAD is indistinguishable against chosen-ciphertext attacks if $\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{AEAD}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

[12] Assuming the above notation for AEAD schemes, we give a similar syntax to AE schemes: an AE scheme encrypts a message m under key k with nonce n to produce a ciphertext denoted $\mathsf{enc}^*(k,n,m)$.

Game $\mathsf{AUTH}(\mathcal{A})$
00 $k \leftarrow_{\$} \mathsf{AEAD.gen}$
01 $C \leftarrow \emptyset, N \leftarrow \emptyset$
02 $\mathcal{A}^{\mathrm{Enc,Dec}}$
03 lose

Oracle $\mathrm{Enc}(n,d,m)$
04 require $n \notin N$
05 $N \xleftarrow{\cup} \{n\}$
06 $c \leftarrow \mathsf{AEAD.enc}(k,n,d,m)$
07 $C \xleftarrow{\cup} \{(n,d,c)\}$
08 return c

Oracle $\mathrm{Dec}(n,d,c)$
09 $m \leftarrow \mathsf{AEAD.dec}(k,n,d,c)$
10 if $m \neq \perp \wedge (n,d,c) \notin C$:
11 win
12 return m

Game $\mathsf{subAUTH}(\mathcal{A})$
00 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
01 $k \leftarrow_{\$} \mathsf{AEAD.gen}_{i_{\mathsf{gen}}}$
02 $C \leftarrow \emptyset, N \leftarrow \emptyset$
03 $\mathcal{A}^{\mathrm{Enc,Dec}}$
04 lose

Oracle $\mathrm{Enc}(n,d,m)$
05 require $n \notin N$
06 $N \xleftarrow{\cup} \{n\}$
07 $c \leftarrow \mathsf{AEAD.enc}_{i_{\mathsf{S}}}(k,n,d,m)$
08 $C \xleftarrow{\cup} \{(n,d,c)\}$
09 return c

Oracle $\mathrm{Dec}(n,d,c)$
10 $m \leftarrow \mathsf{AEAD.dec}_{i_{\mathsf{R}}}(k,n,d,c)$
11 if $m \neq \perp \wedge (n,d,c) \notin C$:
12 win
13 return m

Fig. 4 Games modelling authenticity (AUTH) and subverted authenticity (subAUTH) of an authenticated encryption scheme with associated data AEAD.

5.2 Authenticity

We formalise the authenticity of an AEAD scheme via the game AUTH in Fig. 4 (left). For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD}}(\mathcal{A}) := \Pr[\mathsf{AUTH}(\mathcal{A})]$$

and say that AEAD provides authenticity if $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

5.3 Subverting AEAD

We note that AEAD satisfies the generic syntax introduced above in Section 4, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{AEAD.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{AEAD.enc}$ and receiver algorithm $\Pi.\mathsf{R} = \mathsf{AEAD.dec}$. We may thus apply the generic notions of subversion and undetectability introduced in Section 4.1. In Fig. 3 (right) and Fig. 4 (right) we specify the games subIND-CCA and subAUTH (respectively), modelling the adversary's ability to compromise the expected security properties of a scheme Π when that scheme has been subverted.

We note that for symmetric primitives, key generation is unlikely to be subverted in practice as symmetric keys are typically generated by some external means not connected with or influenced by the scheme itself — e.g. through key agreement protocols[13], or by a trusted platform module. Nevertheless, we retain a syntax that allows for the more general case.

[13] While many different types of security model for key agreement exist (see [52] for a recent overview), all of them have in common that they guarantee that all produced session keys are effectively perfect, i.e., uniformly distributed in some convenient key space.

6 Message Authentication Schemes

Cryptographic message authentication is typically realised with a message authentication code (MAC). Given a key k and a message m, a tag t is deterministically derived as per $t \leftarrow \mathsf{tag}(k,m)$. The (textbook) method to verify the authenticity of m given t is to recompute $t' \leftarrow \mathsf{tag}(k,m)$ and to consider m authentic iff $t' = t$. If this final tag comparison is implemented carelessly, a security issue might emerge: A natural yet naive way to perform the comparison is to check the tag bytes individually in left-to-right order until either a mismatch is spotted or the right-most bytes have successfully been found to match. Note that, if tags are not matching, such an implementation might easily give away, as timing side-channel information, the length of the matching prefix, allowing for practical forgery attacks via step-wise guessing.

This issue is understood by the authors of major cryptographic libraries, which thus contain carefully designed constant-time string comparison code. A consequence is that services for tag generation and verification are routinely split into two separate functions tag and vfy.[14] Our notion of a message authentication scheme follows this approach. It comprises MAC based authentication as a special case, but it also comprises the more exotic randomised MACs as considered in [1].

Formally, a scheme MAC providing message authentication consists of algorithms $\mathsf{MAC.gen}, \mathsf{MAC.tag}, \mathsf{MAC.vfy}$ and associated spaces $\mathcal{K}, \mathcal{M}, \mathcal{T}$. The key generation algorithm MAC.gen outputs a key $k \in \mathcal{K}$. The tagging algorithm MAC.tag takes a key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and returns a message, tag pair $(m,t) \in \mathcal{M} \times \mathcal{T}$. The verification algorithm MAC.vfy takes a key $k \in \mathcal{K}$, a message $m \in \mathcal{M}$, and a tag $t \in \mathcal{T}$, and returns either the message m (indicating that the tag is accepted) or the special symbol $\perp$ to indicate rejection.[15] A shortcut notation for this syntax is

$$\mathsf{MAC.gen} \to \mathcal{K} \quad \text{and} \quad \mathcal{K} \times \mathcal{M} \to \mathsf{MAC.tag} \to \mathcal{M} \times \mathcal{T}$$
$$\text{and} \quad \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \mathsf{MAC.vfy} \to \mathcal{M} \cup \{\perp\}.$$

We formalise the (strong) unforgeability of a message authentication scheme via the game UF in Fig. 5 (left, first column). For any adversary $\mathcal{A}$ we define the advantage $\mathsf{Adv}^{\mathsf{uf}}_{\mathsf{MAC}}(\mathcal{A}) := \Pr[\mathrm{UF}(\mathcal{A})]$ and say that the scheme MAC is (strongly) unforgeable if $\mathsf{Adv}^{\mathsf{uf}}_{\mathsf{MAC}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

6.1 Subverting MACs

We note that MACs satisfy the generic syntax introduced above in Section 4, with key generation algorithm $\Pi.\mathsf{gen} =$

[14] See https://nacl.cr.yp.to/auth.html for an example.

[15] It is more common to consider the output of a MAC verification algorithm to be a bit representing acceptance or rejection; this can be obtained from our syntax by evaluating $[\mathsf{MAC.vfy}(k,m,t) = m]$.

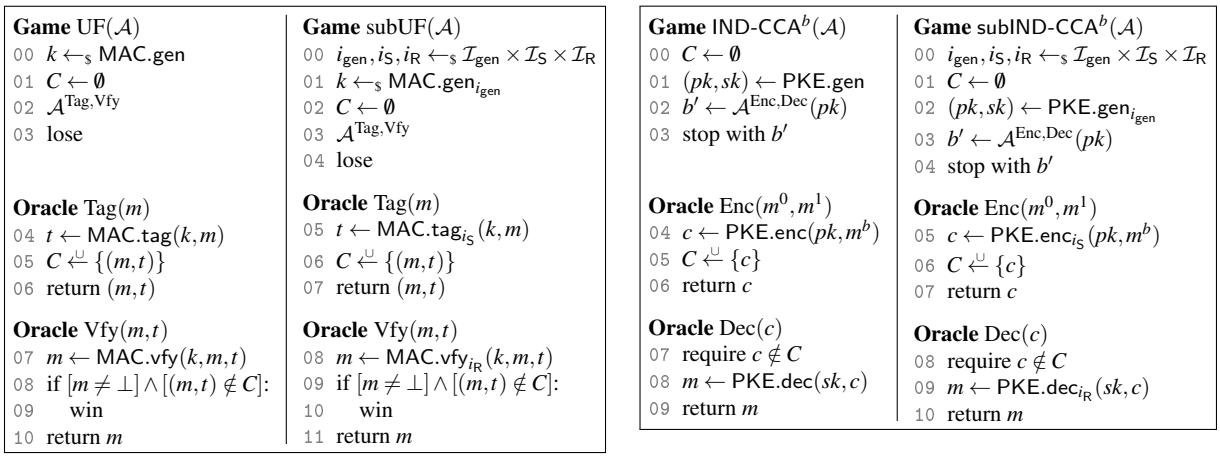

Fig. 5 Left: Games modelling the unforgeability (UF) and subverted unforgeability (subUF) of a message authentication scheme. **Right:** Games modelling indistinguishability under chosen-ciphertext attacks (IND-CCA), and subverted indistinguishability under chosen-ciphertext attacks (subIND-CCA) for a public key encryption scheme PKE.

MAC.gen, sender algorithm $\Pi.\mathsf{S} = \mathsf{MAC.tag}$, receiver algorithm $\Pi.\mathsf{R} = \mathsf{MAC.vfy}$. We may thus apply the generic notions of subversion introduced in Section 4.1. We obtain the notion of subverted unforgeability subUF, as in Fig. 5 (left, second column).

6.1.1 Discussion

We note that for symmetric primitives, key generation is unlikely to be subverted (see the discussion at Section 5.3), leaving us with the possibility that either the tagging or the verification algorithm (or both) could be subverted. However, as tagging and verification are typically performed by distinct, remote parties, successfully conducting such attacks would require replacing implementations of *two* participants, which we think is considerably less feasible than replacing only one implementation.

7 Public Key Encryption Schemes

In this section, we consider ASAs against PKE schemes. A treatment of key encapsulation mechanisms (KEMs) is given in Appendix A.1.

A PKE scheme $\mathsf{PKE} = (\mathsf{PKE.gen}, \mathsf{PKE.enc}, \mathsf{PKE.dec})$ consists of a triple of algorithms together with key spaces $\mathcal{K}_\mathsf{S}, \mathcal{K}_\mathsf{R}$, a message space $\mathcal{M}$ and a ciphertext space $\mathcal{C}$. The key-generation algorithm PKE.gen returns a pair $(pk,sk) \in \mathcal{K}_\mathsf{S} \times \mathcal{K}_\mathsf{R}$ consisting of a public key and a private key. The encryption algorithm PKE.enc takes a public key pk and a message $m \in \mathcal{M}$ to produce a ciphertext $c \in \mathcal{C}$. Finally, the decryption algorithm PKE.dec takes a private key sk and a ciphertext $c \in \mathcal{C}$, and outputs either a message $m \in \mathcal{M}$ or the special symbol $\bot \notin \mathcal{M}$ to indicate rejection. The correctness requirement is that for $(pk,sk) \leftarrow_\$ \mathsf{gen}$ and $m \in \mathcal{M}$ and $c \leftarrow \mathsf{PKE.enc}(pk,m)$ and $m' \leftarrow \mathsf{PKE.dec}(sk,c)$ the probability that $m' \neq m$ is upper-bounded by δ, where the probability is over all coins involved.

We formalise the indistinguishability under chosen-ciphertext attack of a PKE scheme via the game IND-CCA in Fig. 5 (right, first column). For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PKE}}(\mathcal{A}) := |\Pr[\mathsf{IND\text{-}CCA}^0(\mathcal{A})] - \Pr[\mathsf{IND\text{-}CCA}^1(\mathcal{A})]|$$

and say that scheme PKE is indistinguishable against chosen-ciphertext attacks if $\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{PKE}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

7.1 Subverting PKE Schemes

We note that PKE schemes satisfy the generic syntax introduced above in Section 4, with the key generation algorithm $\Pi.\mathsf{gen} = \mathsf{PKE.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{PKE.enc}$, and receiver algorithm $\Pi.\mathsf{R} = \mathsf{PKE.dec}$. We may thus apply the generic notions of subversion introduced in Section 4.1. See Fig. 5 (right, second column) for the game subIND-CCA, modelling the adversary's ability to compromise IND-CCA when interacting with a subverted scheme.

7.1.1 Discussion

For PKE schemes, subverting the encryption algorithm is less interesting, as the sender has no secret information to leak. It would be possible to consider the subversion of encryption with the view of compromising confidentiality of ciphertexts, but we are targeting the stronger notion of key recovery (which will lead to a full compromise of the confidentiality of all ciphertexts). For PKE schemes, in contrast to symmetric encryption, subverting the key generation

algorithm is a meaningful option, and we explain in Section 4.3 how subversion attacks can be amplified when applied together with a subverted key generation algorithm.

8 Concrete Subversion Attacks via Acceptance vs. Rejection

We assume that the objective of a subverted receiver algorithm is to leak a bit string $k_\ell \in \{0,1\}^\lambda$ representing either some leakage that will enable recovery of the secret (private) key k_{R}, following the discussion at Sections 4.2.1 and 4.3, or else a string that is sufficient to break security in the sense of Section 4.4. We refer to k_ℓ as the leakage key in what follows. At worst, from the subverter's perspective, the leakage key will simply be a bit string representation of k_{R}.

We propose two key-recovering subversion attacks against a scheme $\Pi = (\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R})$ satisfying the syntax given in Section 4. While both attacks subvert the receiver algorithm only, they differ in that our first attack is passive (can be mounted by a mass surveillance adversary who eavesdrops) and our second attack is active (requires intercepting and modifying sender outputs in transmission – i.e., ciphertexts, in the case of AEAD or PKE, or message-tag pairs in the MAC case.). The driving principles behind the two attacks are closely related: In both cases the receiver algorithm of the attacked scheme is manipulated such that it marginally deviates from the regular accept/reject behaviour; by making these deviations depend on the leakage key, the bits of the latter are leaked one by one.

Our passive attack rejects a sparse subset of the receiver inputs that the unmodified algorithm would accept. Our active attack does the opposite by accepting certain receiver inputs that the unmodified algorithm would reject. A property of the former (passive) attack is that the scheme's probability of incorrect decryption is increased by a small amount (rendering it detectable with the same probability); we believe however that in settings where rejected messages are automatically retransmitted by the sender (for example, in low-level network encryption like IPSec), this attack is still practical and impactful. Our active attack does not influence correctness. However, as key bits are leaked only when the receiver algorithm is exposed to bogus inputs, successful adversaries are necessarily active. The active attack furthermore has the following attractive property: The underlying receiver outputs (i.e., messages) corresponding to the injected inauthentic receiver inputs are not arbitrary (and thus unexpected to the processing application), but identical with sender inputs previously sent by the sender algorithm. This allows attacks to be kept "under the radar": the receiver will not realise that an attack has been mounted, as all accepted messages it receives will be those sent by the sender.

We note that both of our subversions are stateless, which not only allows for much easier backdoor implementation from a technical perspective but also should decrease the likelihood that an implemented attack is detected through code review or observing memory usage. That said, our passive attack also has a stateful variant with an interesting additional practicality feature. We discuss this further below. We note that our subversion approach, for leaking at most one bit per operation, remains on the conservative side. Depending on the circumstances, in practice, more aggressive methods that leak more than one bit per operation, are expected to be easily derived from our subversion proposals.

Exp $\mathrm{CC}(S,\eta)$
00 $S' \leftarrow \emptyset$; $l \leftarrow 0$
01 while $S' \subsetneq S$:
02 $\quad s \leftarrow_{\$} S$
03 $\quad$ if $B(\eta)$:
04 $\quad\quad S' \xleftarrow{\cup} \{s\}$
05 $\quad l \leftarrow l+1$
06 stop with l

Fig. 6 Coupon collector experiment (see Lemma 8.1). Recall that $B(\eta)$ denotes a Bernoulli trial with success probability η (see Section 3).

8.1 Combinatorics: Coupon Collection

The passive and active attacks both exfiltrate secret key material one bit at a time. The following lemma recalls a standard coupon collector statement that will help analysing the efficiency of this approach, in particular how long it takes until all bits are extracted. For a proof of the lemma, see e.g. [44, §8.4].

Lemma 8.1 *Fix a finite set S (of "coupons") and a probability $0 < \eta \leq 1$. Experiment $\mathrm{CC}(S,\eta)$ in Fig. 6 measures the number of iterations it takes to visit all elements of S ("collect all coupons") when picked uniformly at random and considered with probability η. The expected number of iterations is given by $O(n\log n)$, where $n = |S|$. More precisely we have*

$$\mathbb{E}[\mathrm{CC}(S,\eta)] = \frac{|S|}{\eta}\left(\frac{1}{1}+\frac{1}{2}+\ldots+\frac{1}{|S|}\right) = O(n\log n).$$

Note that parameter η is fully absorbed by the $O(\cdot)$ notation.

8.2 Passive Attack

We first give an intuition of our passive attack. Our attack subverts the receiver algorithm so that an adversary who observes decryption error events in a "normal" run of communication between sender and receiver is able to learn bits of the leakage key. The subverted receiver monitors incoming ciphertexts. It applies a hash function to each of them to obtain a pointer to a bit of the leakage key. It then, with a

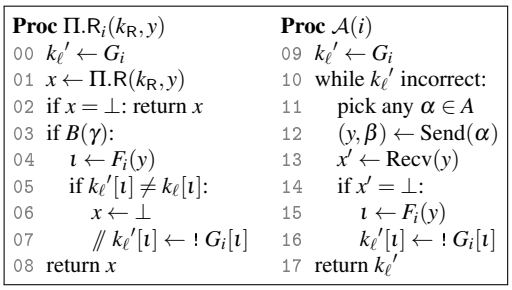

Fig. 7 Passive subversion of receiver algorithm $\Pi.\mathsf{R}$ of a scheme Π. As in Fig. 6, $B(\cdot)$ denotes a Bernoulli trial. We let $!\,b := 1 - b$ denote the inversion of a bit value $b \in \{0,1\}$. **Left:** Decryption subversion as in Section 4.1. Line 07 is redundant if the attack is stateless; in a stateful attack this line is meaningful – see the discussion below. **Right:** Key recovering adversary for game KRP as in Section 4.2.

configurable probability, artificially rejects the ciphertext if the indicated bit of the leakage key does not match some hard-coded reference value. The adversary is able to apply the same hash function to the ciphertext and thus learns whether the bit position deviates from the reference value. By bit-wise learning the difference between the leakage key and the reference value, eventually the adversary can put the complete leakage key together.

In the remaining part of this section, we describe the specification of our subversion and KRP adversary in detail, and analyse their effectiveness.

8.2.1 Description of our Passive Attack

We define our passive subversion of the receiver algorithm $\Pi.\mathsf{R}$ of a scheme Π in Fig. 7 (left). It is parameterised by a probability $0 \leq \gamma \leq 1$, a large index space $\mathcal{I}_\mathsf{R}$, a PRF $(F_i)_{i \in \mathcal{I}_\mathsf{R}}$, and a family $(G_i)_{i \in \mathcal{I}_\mathsf{R}}$ of random constants. For the PRF we require that it be a family of functions $F_i \colon \mathcal{Y} \to [0\,..\,\lambda - 1]$ (that is: a pseudo-random mapping from the ciphertext space to the set of bit positions of a leakage key k_ℓ), and for the constants we require that $G_i \in \{0,1\}^\lambda$ (that is: a random element of the set of leakage keys $\{0,1\}^\lambda$). (That we use the same index space $\mathcal{I}_\mathsf{R}$ for two separate primitives is purely for notational convenience; our analyses will actually assume that (F_i) and (G_i) are independent.[16])

We provide details on our attack. The idea is that k_ℓ' (line 00), which is shared by the subverted algorithm and the key recovering adversary through knowledge of G_i, represents an initial reference key[17]; the key recovery adversary, throughout the attack, learns the bits that differ between k_ℓ and k_ℓ'. This means that the subversion only needs to leak (on average) half as many bits compared to leaking the whole of k_ℓ. The Bernoulli trial (line 03) controls the rate with which such differing bits are exfiltrated, and the PRF (line 04) controls which bit position ι is affected in each iteration. By PRF security, these bit positions can be assumed uniformly distributed (though knowledge of the subversion index i allows tracing which ciphertext is mapped to which position). Any bit difference is communicated to the adversary by artificially rejecting (line 06) the ciphertext, although it is actually valid.

We specify a corresponding KRP adversary in Fig. 7 (right). It starts with the same random string G_i as the subversion and traces the bit updates of $\Pi.\mathsf{R}_i$ until eventually the full key k_ℓ is reconstructed. We assume that $\mathcal{A}$ can tell whether the full leakage key k_ℓ has been recovered (line 10), e.g. by recovering the secret key k_R from k_ℓ and verifying one or more recorded authentic outputs with it.

Note that our adversary $\mathcal{A}$ does not need to know the sender inputs (equivalently, receiver outputs) $x \in \mathcal{X}$, which typically represent plaintext messages in the settings we consider, emerging throughout the experiment: The core of the attack, in lines 13 to 16, is independent of the value of x. This considerably adds to the practicality of our attack: While messages are not always secret information, *in practice* they might be hard to obtain. Conducting mass-surveillance attacks is certainly easier if the attacks depend exclusively on the knowledge of ciphertexts (like in our case, line 15).

While we present our subversion as stateless (i.e., the reference key is kept static between invocations), it also works if the $\Pi.\mathsf{R}_i$ algorithm maintains state between any two invocations and remembers which differing bit positions have already been communicated. Activate line 07, and execute line 00 only during the first invocation, to obtain the stateful attack. With respect to the detectability and key recovery notions from Section 4, the attack's performance is the same whether the subversion is stateful or not. The stateful version offers better correctness *after* a subversion is detected, in the sense that the algorithm will only behave unexpectedly at most $|k_\ell| = \lambda$ occasions; once the leakage key k_ℓ has been exfiltrated, the subverted scheme $\Pi.\mathsf{R}_i$ behaves identically to the honest scheme $\Pi.\mathsf{R}$. (This case is practically less relevant and not covered by our formal models.)

We establish the following statements about the key recoverability and undetectability of our passive subversion attack.

Theorem 8.1 *For a δ-correct scheme Π, let $\Pi.\mathsf{R}_i$ be defined as in Fig. 7 (left) and $\mathcal{A}$ as defined in Fig. 7 (right). If F_i behaves like a random function and constants G_i are uniformly distributed, then for any message sampler* MS*, the key recovery advantage $\mathsf{Adv}^{\mathrm{krp}}_{\mathrm{MS}}(\mathcal{A})$ is expected to reach value 1 once the receive algorithm was invoked on $O(\lambda \log \lambda)$ different inputs.*

[16] At the expense of introducing more symbols we could also have formally separated the index spaces of (F) and (G). We believe that our concise notation adds significantly to readability.

[17] For generality, we consider G a keyed family of constants. The simplest case would have all constants fixed to the same hardcoded string (say, the string of all zeroes), which would aid in reducing the size of the implementation code.

Proof We model algorithm $\mathcal{A}(i)$ by experiment $\mathrm{CC}(S,\eta)$ from Fig. 6, with $S=[0..\lambda-1]$ and $\eta=1-(\delta-1)(\gamma/2-1)$. The (pseudo-)randomness of F_i ensures that elements of $s\in S$, here representing the possible values of the index ι (line 04), are picked uniformly at random. The probability $\eta=1-(\delta-1)(\gamma/2-1)=\delta+(1-\delta)(\gamma/2)$ arises through success of the CC experiment being equivalent to the $\Pi.\mathsf{R}_i$ outputting $x=\perp$. This occurs either:

- Through an early exit with $\perp=\Pi.\mathsf{R}(k_\mathsf{R},y)$ at line 02, which has probability δ.
- Else, continuing to line 03 with $\perp\neq\Pi.\mathsf{R}(k_\mathsf{R},y)$ and triggering both line 03 (with probability γ) and line 05 (with probability $1/2$, as $\Pr[k_\ell{}'[\iota]\neq k_\ell[\iota]]$ for $\iota\leftarrow F_i(t)$ is $1/2$).

Applying Lemma 8.1 gives the expected number of messages to be sent as $O(\lambda\log\lambda)$.

Theorem 8.2 *Let $\mathcal{A}$ be an adversary playing the* UDR *game (as in Fig. 1, right), such that $\mathcal{A}$ makes at most q queries to the receiver oracle* Recv. *The undetectability advantage of the subversion $\Pi.\mathsf{R}_i$, as defined in Fig. 7 (left), is bounded by*

$$\mathsf{Adv}^{\mathrm{UDR}}_{\Pi}(\mathcal{A})\leq 1-(1-\gamma)^q.$$

Proof Any adversary playing the UDR game against the subverted $\Pi.\mathsf{R}_i$ must, in order to win, trigger $x=\perp$ with a valid sender output (receiver input) y. More precisely, the adversary $\mathcal{A}$ must find y such that $\Pi.\mathsf{R}(k_\mathsf{R},y)\neq\perp$ but $\Pi.\mathsf{R}_i(k_\mathsf{R},y)=\perp$. Figure 8 (left) shows the (obviously) best adversarial strategy. Even if the adversary can submit y such that $\iota\leftarrow F_i(y)$ would be assigned in line 04 (Fig. 7), this is contingent on $B(\gamma)$ succeeding in line 03; thus $\Pr[x=\perp]\leq\gamma$ in line 05 (Fig. 8). Clearly, detection adversary $\mathcal{A}$ (Fig. 8, left) always returns 1 when interacting with the unsubverted receiver algorithm, as always $x\neq\perp$. Thus,

$$\begin{aligned}\mathsf{Adv}^{\mathrm{UDR}}_{\Pi}(\mathcal{A})&=|\Pr[\mathrm{UDR}^1(\mathcal{A})]-\Pr[\mathrm{UDR}^0(\mathcal{A})]|\\&\leq 1-(1-\gamma)^q.\end{aligned}$$

8.3 Active Attack

In this section we describe our second subversion attack. In contrast to the previous attack, key recovery requires an active adversary, i.e., one who injects crafted ciphertexts into the regular transmission stream. Our ASA has the desirable property (from the point of view of the subverter) that correctness is maintained.

We give an overview of our attack for the case of AEAD. (The generalisation to MAC and PKE is immediate; for the generic version following our abstract syntax see Fig. 9.) A prerequisite of the attack is a keyed random permutation P_i of the AEAD ciphertext space. The key is known exclusively to the subversion adversary. The AEAD encryption algorithm S remains unmodified. For honestly generated ciphertexts, the (subverted) algorithm R_i implements the unmodified AEAD decryption routine R. This ensures correctness.

To start a key recovery attack, the subversion adversary waits for an honest ciphertext c and replaces it with $P_i(c)$. That is, the adversary suppresses the delivery of c and instead injects a "randomised same-length version" of the ciphertext. By the authenticity property of the AEAD scheme, the unmodified R algorithm would reject this ciphertext. This is where the (subverted) R_i deviates from R: If any incoming ciphertext is deemed invalid upon decryption with R, with the expectation that this could be the case due to a KRA attack being in operation, R_i applies P_i^{-1} to c and tries to decrypt the result (with R). If R rejects, R_i rejects also (interpretation: c was simply a random ciphertext, not injected by the KRA adversary). If however R accepts, then R_i concludes that a KRA attack is in operation, and that it (R_i) is supposed to leak key material.[18] Observing that R_i just recovered the originally encrypted message m, we let R_i either deliver that message, or we let it return $\perp$, i.e., indicate decryption failure. That way, if a message is delivered by R_i, it is always correct. As in Sec 8.2, we modulate key bits into the decision of delivering vs rejecting.

The above should make clear how the attack works. Details, and the generic version, are in Section 8.3.2. We note that a technical prerequisite of the attack is that for valid ciphertexts c, $P_i(c)$ should not also be a valid ciphertext. As P is a random permutation, the standard AUTH and UF properties of AEAD and MAC ensure this. However, the situation is different for PKE where it is easy to define schemes that accept every ciphertext input, e.g. by outputting a valid but dummy message. We resolve this technicality by requir-

Proc $\mathcal{A}$
00 $(k_\mathsf{S},k_\mathsf{R})\leftarrow_\$\Pi.\mathsf{gen}$
01 repeat q times:
02 pick any $x\in\mathcal{X}$
03 $y\leftarrow\Pi.\mathsf{S}(k_\mathsf{S},x)$
04 $x\leftarrow\mathrm{Recv}(k_\mathsf{S},y)$
05 if $x=\perp$:
06 return 0
07 return 1

Proc $\mathcal{A}$
00 $(k_\mathsf{S},k_\mathsf{R})\leftarrow_\$\Pi.\mathsf{gen}$
01 $S\leftarrow\{y\}$
02 $\mathsf{ct}=0$
03 while $\mathsf{ct}<q$:
04 pick any $x\in\mathcal{X}$
05 $y\leftarrow\Pi.\mathsf{S}(k_\mathsf{S},x)$
06 $y'\leftarrow_\$\mathcal{Y}\setminus S$
07 if $\Pi.\mathsf{R}(y')=\perp$:
08 $\mathsf{ct}\leftarrow\mathsf{ct}+1$
09 $x\leftarrow\mathrm{Recv}(k_\mathsf{S},y')$
10 $S\xleftarrow{\cup}\{y'\}$
11 if $x\neq\perp$:
12 return 0
13 return 1

Fig. 8 Detection adversaries for Game UDR as in Fig. 1. **Left:** For the passive attack from Fig. 7. **Right:** For the active attack from Fig. 9.

[18] Multiple options for this exist, for instance could the full k_R be embedded into the returned m; this would be powerful, but also has practical disadvantages that would hinder effectiveness. We hence pursue a different, milder approach.

ing the mild assumption of ciphertext sparseness: We say that a PKE scheme has a sparse ciphertext space if the decryption algorithm makes an internal decision about the validity of an incoming ciphertext, with the property that uniformly picked ciphertexts are deemed invalid with overwhelming probability.

We studied a range of practically relevant PKE schemes and observe that all of them satisfy the ciphertext sparseness demand. For reference we provide corresponding details for OAEP and Cramer-Shoup encryption in Appendix B. We further observe that the general classes of plaintext-aware schemes [13] (see also Appendix C) and of schemes with publicly verifiable ciphertexts [50] have this property as well. We confirm also for all four of the NIST post-quantum cryptography round 3 finalists[19], that are specifically designed to mask decryption failures by accepting every ciphertext and outputting a random value rather than the rejection symbol, that they provide ciphertext sparseness: The mechanics of the decryption/decapsulation algorithms are such that first an internal yet explicit ciphertext validity decision is made and then either the correct or an independent, randomised value is output. Our subversions can easily adapt to such specifications and be directly based on the outcome of the validity check.

Lastly, we note that for the active attack there is no advantage to the subverted receiver keeping state. This is because the subverted receiver reveals key bits only when explicitly queried by the adversary – thus, the adversary is able to maintain all necessary state. Note this is in contrast to our passive attack where the adversary observes the receiver but does not interact with it, and ultimately thus the attack could benefit mildly from the subversion R_i keeping state.

In the remaining part of this section, we define ciphertext sparseness, describe the specification of our subversion and KRA adversary, and analyse their effectiveness.

8.3.1 Ciphertext Sparseness

We define ciphertext sparseness for a scheme Π as follows: We say that $\Pi = (\Pi.\mathsf{gen}, \Pi.\mathsf{S}, \Pi.\mathsf{R})$ is ciphertext ε-sparse if $\Pr[\mathsf{R}(y) \neq \perp] \leq \varepsilon$ for $y \leftarrow_{\$} \mathcal{Y}$. If ε is negligibly small, we refer to the scheme as being ciphertext sparse. For AEAD and MAC schemes, ciphertext sparseness is a corollary of the unforgeability (authenticity) properties. In particular: an AEAD scheme that has $\mathsf{Adv}^{\mathsf{auth}}_{\mathsf{AEAD}}(\mathcal{A}) \leq \varepsilon$ is ciphertext ε-sparse, as if $\Pr[\mathsf{R}(y) \neq \perp] > \varepsilon$ for $y \leftarrow_{\$} \mathcal{Y}$, then an adversary who simply chooses an element of $\mathcal{Y}$ uniformly at random would win the AUTH game with probability greater than ε. Similarly, a MAC scheme with $\mathsf{Adv}^{\mathsf{uf}}_{\mathsf{MAC}}(\mathcal{A}) \leq \varepsilon$ is ciphertext ε-sparse. As discussed above, ciphertext sparseness is a reasonable assumption also for many practical PKE schemes.

[19] Classic McEliece, CRYSTALS-KYBER, NTRU and SABER. Detailed information on the submissions, in particular their specifications, are available at the NIST PQC website https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions

Proc $\Pi.\mathsf{R}_i(k_\mathsf{R}, y)$
00 $k_\ell' \leftarrow G_i$
01 $x \leftarrow \Pi.\mathsf{R}(k_\mathsf{R}, y)$
02 if $x \neq \perp$:
03 return x
04 $y' \leftarrow P_i^{-1}(y)$
05 $x' \leftarrow \Pi.\mathsf{R}(k_\mathsf{R}, y')$
06 if $x' = \perp$:
07 return $\perp$
08 $\iota \leftarrow F_i(y')$
09 if $k_\ell[\iota] \neq k_\ell'[\iota]$:
10 return x'
11 return $\perp$

Proc $\mathcal{A}(i)$
12 $k_\ell' \leftarrow G_i$
13 while k_ℓ' incorrect:
14 pick any α
15 $(y, \beta) \leftarrow \mathrm{Send}(\alpha)$
16 $y' \leftarrow P_i(y)$
17 if $y' = y$, jump to line 14
18 $x \leftarrow \mathrm{Recv}(y')$
19 if $x \neq \perp$:
20 $\iota \leftarrow F_i(y')$
21 $k_\ell'[\iota] \leftarrow !\, G_i[\iota]$
22 return k_ℓ'

Fig. 9 Active subversion of the receiver algorithm $\Pi.\mathsf{R}$ of a ciphertext ε-sparse scheme Π (see the discussion at Section 8.3). **Left:** Decryption subversion as in Section 4.1. **Right:** Key recovering adversary for game KRA as in Section 4.2. The adversary needs to have no influence over messages (modelled by α; see the discussion at Section 4.2). As before we let ! denote the inversion of a bit value.

8.3.2 Description of our Active Attack

We define our active subversion of the receiver algorithm of ciphertext-sparse scheme Π in Fig. 9 (left). It is parameterised by a large index space $\mathcal{I}_\mathsf{R}$, a PRF $(F_i)_{i \in \mathcal{I}_\mathsf{R}}$, a PRP $(P_i)_{i \in \mathcal{I}_\mathsf{R}}$, and a family $(G_i)_{i \in \mathcal{I}_\mathsf{R}}$ of random constants. (As in Section 8.2, our analyses will assume that (F_i) and (P_i) and (G_i) are independent.) For the PRF we require that it be a family of functions $F_i \colon \mathcal{Y} \to [0\,..\,\lambda - 1]$ (that is: a pseudo-random mapping from the output space to the set of bit positions of a leakage key $k_\ell \in \{0,1\}^\lambda$), for the PRP we require that it be a family of length-preserving permutations $P_i \colon \mathcal{Y} \to \mathcal{Y}$ (that is: a pseudo-random bijection on the sender output space), and for the constants we require that $G_i \in \{0,1\}^\lambda$ (that is: a random element of the set of leakage keys).

The idea of our attack is as follows. Lines 01 to 03 of $\Pi.\mathsf{R}_i$ ensure that authentic receiver inputs are always accepted (no limitation on correctness). If however a receiver input (sender output) y is identified as not valid, i.e., is unauthentic, then a secret further check is performed: The original value y is mapped to an unrelated value y' using the random permutation (line 04), and the result y' is checked for validity (line 05). For standard (invalid) sender outputs y this second validity check should also fail, and in this case algorithm $\Pi.\mathsf{R}_i$ rejects as expected (lines 06 and 07). The normally not attainable case that the second validity check succeeds is used to leak key bits. The mechanism for this (lines 08 to 11) is as in our passive attack from Section 8.2, namely by communicating via accept/reject decisions the positions where the bits of a hard-coded random reference value k_ℓ' and the to-be-leaked key k_ℓ differ.

The corresponding key recovery adversary crafts these required bogus receiver inputs by obtaining a valid sender

output[20] y (line 15) and modifies it in line 16. The information thus leaked by the validity checking routine is used to reconstruct target leakage key k_ℓ in the obvious way (lines 19 to 21). We establish the following statements about the key recoverability and undetectability of our active subversion attack.

Theorem 8.3 *For an ε-sparse scheme Π, let $\Pi.\mathsf{R}_i$ be defined as in Fig. 9 (left) and $\mathcal{A}$ as in Fig. 9 (right). If F_i and P_i behave like random functions, and constants G_i are uniformly distributed, then for any message sampler* MS*, the key recovery advantage $\mathsf{Adv}^{\mathrm{kra}}_{\mathrm{MS}}(\mathcal{A})$ is expected to reach value* 1 *once the receive algorithm was invoked on $O(\lambda \log \lambda)$ different inputs.*

Proof By the ciphertext ε-sparseness of the scheme Π, each invocation of algorithm $\Pi.\mathsf{R}_i$ in an execution of attack $\mathcal{A}(i)$ has $x \neq \bot$ in line 02 (Fig. 9) with probability ε and thus line 04 is reached with probability $1-\varepsilon$. We model algorithm $\mathcal{A}(i)$ by the experiment $\mathrm{CC}(S,\eta)$ from Fig. 6, with $S = [0\,..\,\lambda-1]$ and $\eta = (1-\varepsilon)/2$. The (pseudo-)randomness of F_i ensures that elements of $s \in S$, here representing the possible values of the index ι (line 20), are picked uniformly at random. The probability $1/2$ arises through success of the CC experiment being equivalent to the condition $x \neq \bot$ in line 19. This occurs precisely when $\Pi.\mathsf{R}_i$ returns $x' \neq \bot$ in line 10, which is conditional on $\Pi.\mathsf{R}_i$ reaching past line 07. The probability that $x \neq \bot$ in line 19 is $1/2$ as this is the probability that for any sender output y' and $\iota \leftarrow F_i(y'), k_\ell[\iota] \neq G_i[\iota]$ (line 09). We now apply Lemma 8.1, which gives us that the expected number of messages to be sent is $O(\lambda \log \lambda)$.

Theorem 8.4 *Let $\mathcal{A}$ be an adversary playing the* UDR *game (as in Fig. 1, right), such that $\mathcal{A}$ makes at most q queries to the verification oracle* Recv*. If P_i behaves like a random function, and the scheme Π is ciphertext ε-sparse, then the undetectability advantage of the subversion $\Pi.\mathsf{R}_i$, as defined in Fig. 9 (left), is given by $\mathsf{Adv}^{\mathrm{udr}}_{\Pi}(\mathcal{A}) \leq 1-(1-\varepsilon)^q$.*

Proof Any detection adversary $\mathcal{A}$ playing the UDR game against the subverted $\Pi.\mathsf{R}_i$ must, in order to win, trigger $\Pi.\mathsf{R}_i(y) \neq \bot$ with a bogus y. That is, a sender output y with $\Pi.\mathsf{R}(k_\mathsf{R}, y) = \bot$ but $\Pi.\mathsf{R}_i(k_\mathsf{R}, y) \neq \bot$. This will occur if $y = P_i(y')$, where $\Pi.\mathsf{R}(k_\mathsf{R}, y') \neq \bot$. As i is chosen uniformly randomly from $\mathcal{I}_\mathsf{R}$ and P is a (pseudo-)random function, the optimal strategy is to sample values of y' and test whether $\mathrm{Recv}(k_\mathsf{R}, y') \neq \bot$. Algorithm $\mathcal{A}$ in Fig. 8 (right) shows this strategy. When $\mathcal{A}$ interacts with the unsubverted receiver algorithm, we have that $\Pr[\mathrm{UDR}^1(\mathcal{A})] = 1$ by construction. When interacting with the subverted receiver algorithm, $\mathcal{A}$ returns $x \neq \bot$ by either triggering line 02 or line 10 of $\Pi.\mathsf{R}_i$. By the ciphertext sparseness of the scheme, line 02 is triggered with probability $1-\varepsilon$. Triggering line 10 happens with probability $\leq \varepsilon$. Thus we have

$$\mathsf{Adv}^{\mathrm{UDR}}_{\Pi}(\mathcal{A}) = |\Pr[\mathrm{UDR}^1(\mathcal{A})] - \Pr[\mathrm{UDR}^0(\mathcal{A})]| \leq 1-(1-\varepsilon)^q\,.$$

[20] Note that in the symmetric case, such authentic outputs are obtained by intercepting valid communications between the sender and receiver; in the public key case, an adversary can easily craft their own authentic outputs using the public key. We consider adversaries that have no influence on message choices for the most powerful attack (hence the arbitrary value of α in line 14); adversaries who are able to utilise α (and β) may be even more effective.

9 Conclusion

Our work examines subversion attacks against the receiving party, a class of ASA that was missed by previous work. We give an abstract framework and show that ASAs targeting receivers apply equally to any primitive meeting the syntax – namely, AEAD, MAC and PKE schemes. The internal details of our attacks (described in Sections 8.3.2 and 8.2.1) are such that we require a PRF $(F_i)_{i\in\mathcal{I}_\mathsf{R}}$ to uniformly hash ciphertexts to bit positions. In the AEAD setting, this requirement can be dropped where the AEAD scheme meets the widespread design goal of IND$ security [54], i.e. ciphertexts indistinguishable from random bits. Combined with the fact that symmetric keys are typically 256 bits, the first 8 bits of the (uniformly distributed) ciphertexts are sufficient to point to the bit position. This allows for a reduced footprint (and thus significantly adds to the practicability of the attack for an adversary).

Acknowledgements The research of Armour was supported by the EPSRC and the UK government as part of the Centre for Doctoral Training in Cyber Security at Royal Holloway, University of London (EP/P009301/1).

References

1. Fatema Al Mansoori, Joonsang Baek, and Khaled Salah. Subverting MAC: How authentication in mobile environment can be undermined. In *2016 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 870–874, April 2016. doi:10.1109/INFCOMW.2016.7562200.
2. Marcel Armour and Carlos Cid. Partition oracles from weak key forgeries. Cryptology ePrint Archive, Report 2021/1296, 2021. https://eprint.iacr.org/2021/1296.
3. Marcel Armour and Carlos Cid. Partition oracles from weak key forgeries. In Mauro Conti, Marc Stevens, and Stephan Krenn, editors, *Cryptology and Network Security*, pages 42–62, Cham, 2021. Springer International Publishing.
4. Marcel Armour and Bertram Poettering. Substitution attacks against message authentication. *IACR Transactions on Symmetric Cryptology*, 2019(3):152–168, 2019. doi:10.13154/tosc.v2019.i3.152-168.
5. Marcel Armour and Bertram Poettering. Subverting decryption in AEAD. In Martin Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *Lecture Notes in Computer Science*, pages 22–41. Springer, Heidelberg, December 2019. doi:10.1007/978-3-030-35199-1_2.

6. Giuseppe Ateniese, Bernardo Magri, and Daniele Venturi. Subversion-resilient signature schemes. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 364–375. ACM Press, October 2015. doi:10.1145/2810103.2813635.
7. Nimrod Aviram, Benjamin Dowling, Ilan Komargodski, Kenneth G. Paterson, Eyal Ronen, and Eylon Yogev. Practical (post-quantum) key combiners from one-wayness and applications to TLS. Cryptology ePrint Archive, Report 2022/065, 2022. https://eprint.iacr.org/2022/065.
8. Elaine Barker. Nist special publication 800-57 part 1, revision 5. *Recommendation for Key Management*, 2020. doi:10.6028/NIST.SP.800-57pt1r5.
9. Balthazar Bauer, Pooya Farshim, and Sogol Mazaheri. Combiners for backdoored random oracles. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 272–302. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96881-0_10.
10. Mihir Bellare and Viet Tung Hoang. Resisting randomness subversion: Fast deterministic and hedged public-key encryption in the standard model. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 627–656. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_21.
11. Mihir Bellare, Joseph Jaeger, and Daniel Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In Indrajit Ray, Ninghui Li, and Christopher Kruegel, editors, *ACM CCS 2015: 22nd Conference on Computer and Communications Security*, pages 1431–1440. ACM Press, October 2015. doi:10.1145/2810103.2813681.
12. Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-key symmetric encryption: Resisting key exfiltration. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 373–402. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_14.
13. Mihir Bellare and Adriana Palacio. Towards plaintext-aware public-key encryption without random oracles. In Pil Joong Lee, editor, *Advances in Cryptology – ASIACRYPT 2004*, volume 3329 of *Lecture Notes in Computer Science*, pages 48–62. Springer, Heidelberg, December 2004. doi:10.1007/978-3-540-30539-2_4.
14. Mihir Bellare, Kenneth G. Paterson, and Phillip Rogaway. Security of symmetric encryption against mass surveillance. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014, Part I*, volume 8616 of *Lecture Notes in Computer Science*, pages 1–19. Springer, Heidelberg, August 2014. doi:10.1007/978-3-662-44371-2_1.
15. Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption. In Alfredo De Santis, editor, *Advances in Cryptology – EUROCRYPT'94*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer, Heidelberg, May 1995. doi:10.1007/BFb0053428.
16. Pascal Bemmann, Rongmao Chen, and Tibor Jager. Subversion-resilient public key encryption with practical watchdogs. In Juan Garay, editor, *PKC 2021: 24th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 12710 of *Lecture Notes in Computer Science*, pages 627–658. Springer, Heidelberg, May 2021. doi:10.1007/978-3-030-75245-3_23.
17. Sebastian Berndt and Maciej Liskiewicz. Algorithm substitution attacks from a steganographic perspective. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 1649–1660. ACM Press, October / November 2017. doi:10.1145/3133956.3133981.
18. Sebastian Berndt, Jan Wichelmann, Claudius Pott, Tim-Henrik Traving, and Thomas Eisenbarth. ASAP: Algorithm substitution attacks on cryptographic protocols. Cryptology ePrint Archive, Report 2020/1452, 2020. https://eprint.iacr.org/2020/1452.
19. Swarup Bhunia, Michael S Hsiao, Mainak Banga, and Seetharam Narasimhan. Hardware trojan attacks: threat analysis and countermeasures. *Proceedings of the IEEE*, 102(8):1229–1247, 2014.
20. James Birkett and Alexander W. Dent. Relations among notions of plaintext awareness. In Ronald Cramer, editor, *PKC 2008: 11th International Workshop on Theory and Practice in Public Key Cryptography*, volume 4939 of *Lecture Notes in Computer Science*, pages 47–64. Springer, Heidelberg, March 2008. doi:10.1007/978-3-540-78440-1_4.
21. Angèle Bossuat, Xavier Bultel, Pierre-Alain Fouque, Cristina Onete, and Thyla van der Merwe. Designing reverse firewalls for the real world. In Liqun Chen, Ninghui Li, Kaitai Liang, and Steve A. Schneider, editors, *ESORICS 2020: 25th European Symposium on Research in Computer Security, Part I*, volume 12308 of *Lecture Notes in Computer Science*, pages 193–213. Springer, Heidelberg, September 2020. doi:10.1007/978-3-030-58951-6_10.
22. Jan Camenisch, Manu Drijvers, and Anja Lehmann. Anonymous attestation with subverted TPMs. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 427–461. Springer, Heidelberg, August 2017. doi:10.1007/978-3-319-63697-9_15.
23. Rongmao Chen, Xinyi Huang, and Moti Yung. Subvert KEM to break DEM: Practical algorithm-substitution attacks on public-key encryption. In Shiho Moriai and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2020, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 98–128. Springer, Heidelberg, December 2020. doi:10.1007/978-3-030-64834-3_4.
24. Don Coppersmith. Finding a small root of a bivariate integer equation; factoring with high bits known. In Ueli M. Maurer, editor, *Advances in Cryptology – EUROCRYPT'96*, volume 1070 of *Lecture Notes in Computer Science*, pages 178–189. Springer, Heidelberg, May 1996. doi:10.1007/3-540-68339-9_16.
25. Ronald Cramer and Victor Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In Hugo Krawczyk, editor, *Advances in Cryptology – CRYPTO'98*, volume 1462 of *Lecture Notes in Computer Science*, pages 13–25. Springer, Heidelberg, August 1998. doi:10.1007/BFb0055717.
26. Ronald Cramer and Victor Shoup. Design and analysis of practical public-key encryption schemes secure against adaptive chosen ciphertext attack. *SIAM Journal on Computing*, 33(1):167–226, 2003.
27. Claude Crépeau and Alain Slakmon. Simple backdoors for RSA key generation. In Marc Joye, editor, *Topics in Cryptology – CT-RSA 2003*, volume 2612 of *Lecture Notes in Computer Science*, pages 403–416. Springer, Heidelberg, April 2003. doi:10.1007/3-540-36563-X_28.
28. Luca De Feo, Bertram Poettering, and Alessandro Sorniotti. On the (in)security of ElGamal in OpenPGP. In Giovanni Vigna and Elaine Shi, editors, *ACM CCS 2021: 28th Conference on Computer and Communications Security*, pages 2066–2080. ACM Press, November 2021. doi:10.1145/3460120.3485257.
29. Jean Paul Degabriele, Pooya Farshim, and Bertram Poettering. A more cautious approach to security against mass surveillance. In Gregor Leander, editor, *Fast Software Encryption – FSE 2015*, volume 9054 of *Lecture Notes in Computer Science*,

pages 579–598. Springer, Heidelberg, March 2015. doi:10.1007/978-3-662-48116-5_28.

30. Jean Paul Degabriele, Kenneth G. Paterson, Jacob C. N. Schuldt, and Joanne Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 403–432. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_15.
31. Alexander W. Dent. The Cramer-Shoup encryption scheme is plaintext aware in the standard model. In Serge Vaudenay, editor, *Advances in Cryptology – EUROCRYPT 2006*, volume 4004 of *Lecture Notes in Computer Science*, pages 289–307. Springer, Heidelberg, May / June 2006. doi:10.1007/11761679_18.
32. Yevgeniy Dodis, Pooya Farshim, Sogol Mazaheri, and Stefano Tessaro. Towards defeating backdoored random oracles: Indifferentiability with bounded adaptivity. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020: 18th Theory of Cryptography Conference, Part III*, volume 12552 of *Lecture Notes in Computer Science*, pages 241–273. Springer, Heidelberg, November 2020. doi:10.1007/978-3-030-64381-2_9.
33. Yevgeniy Dodis, Chaya Ganesh, Alexander Golovnev, Ari Juels, and Thomas Ristenpart. A formal treatment of backdoored pseudorandom generators. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 101–126. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46800-5_5.
34. Yevgeniy Dodis, Ilya Mironov, and Noah Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology – CRYPTO 2016, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 341–372. Springer, Heidelberg, August 2016. doi:10.1007/978-3-662-53018-4_13.
35. Morris J. Dworkin. SP 800-38D: Recommendation for block cipher modes of operation: Galois/Counter Mode (GCM) and GMAC. US National Institute of Standards and Technology, 2007. doi:10.6028/NIST.SP.800-38D.
36. Stefan Dziembowski, Sebastian Faust, and François-Xavier Standaert. Private circuits III: Hardware trojan-resilience via testing amplification. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 142–153. ACM Press, October 2016. doi:10.1145/2976749.2978419.
37. Marc Fischlin, Christian Janson, and Sogol Mazaheri. Backdoored hash functions: Immunizing HMAC and HKDF. In Steve Chong and Stephanie Delaune, editors, *CSF 2018: IEEE 31st Computer Security Foundations Symposium*, pages 105–118. IEEE Computer Society Press, 2018. doi:10.1109/CSF.2018.00015.
38. Marc Fischlin and Sogol Mazaheri. Self-guarding cryptographic protocols against algorithm substitution attacks. In Steve Chong and Stephanie Delaune, editors, *CSF 2018: IEEE 31st Computer Security Foundations Symposium*, pages 76–90. IEEE Computer Society Press, 2018. doi:10.1109/CSF.2018.00013.
39. Federico Giacon, Felix Heuer, and Bertram Poettering. KEM combiners. In Michel Abdalla and Ricardo Dahab, editors, *PKC 2018: 21st International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 10769 of *Lecture Notes in Computer Science*, pages 190–218. Springer, Heidelberg, March 2018. doi:10.1007/978-3-319-76578-5_7.
40. Eu-Jin Goh, Dan Boneh, Benny Pinkas, and Philippe Golle. The design and implementation of protocol-based hidden key recovery. In Colin Boyd and Wenbo Mao, editors, *ISC 2003: 6th International Conference on Information Security*, volume 2851 of *Lecture Notes in Computer Science*, pages 165–179. Springer, Heidelberg, October 2003.
41. Dieter Gollmann. *Computer Security (3. ed.)*. Wiley, 2011. URL: http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1118801326.html.
42. Philip Hodges and Douglas Stebila. Algorithm substitution attacks: State reset detection and asymmetric modifications. *IACR Transactions on Symmetric Cryptology*, 2021(2):389–422, 2021. doi:10.46586/tosc.v2021.i2.389-422.
43. Akiko Inoue, Tetsu Iwata, Kazuhiko Minematsu, and Bertram Poettering. Cryptanalysis of OCB2: Attacks on authenticity and confidentiality. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 3–31. Springer, Heidelberg, August 2019. doi:10.1007/978-3-030-26948-7_1.
44. Richard Isaac. *The pleasures of probability*. Springer Science & Business Media, 2013.
45. Lars R. Knudsen and Tadayoshi Kohno. Analysis of RMAC. In Thomas Johansson, editor, *Fast Software Encryption – FSE 2003*, volume 2887 of *Lecture Notes in Computer Science*, pages 182–191. Springer, Heidelberg, February 2003. doi:10.1007/978-3-540-39887-5_14.
46. Ted Krovetz and Phillip Rogaway. The OCB authenticated-encryption algorithm, 2014. https://tools.ietf.org/html/rfc7253.
47. Julia Len, Paul Grubbs, and Thomas Ristenpart. Partitioning oracle attacks. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021: 30th USENIX Security Symposium*, pages 195–212. USENIX Association, August 2021.
48. Hui Ma, Rui Zhang, Guomin Yang, Zishuai Song, Shuzhou Sun, and Yuting Xiao. Concessive online/offline attribute based encryption with cryptographic reverse firewalls - secure and efficient fine-grained access control on corrupted machines. In Javier López, Jianying Zhou, and Miguel Soriano, editors, *ESORICS 2018: 23rd European Symposium on Research in Computer Security, Part II*, volume 11099 of *Lecture Notes in Computer Science*, pages 507–526. Springer, Heidelberg, September 2018. doi:10.1007/978-3-319-98989-1_25.
49. Ilya Mironov and Noah Stephens-Davidowitz. Cryptographic reverse firewalls. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 657–686. Springer, Heidelberg, April 2015. doi:10.1007/978-3-662-46803-6_22.
50. Juan Manuel González Nieto, Mark Manulis, Bertram Poettering, Jothi Rangasamy, and Douglas Stebila. Publicly verifiable ciphertexts. *J. Comput. Secur.*, 21(5):749–778, 2013. doi:10.3233/JCS-130473.
51. Bertram Poettering and Paul Rösler. Combiners for AEAD. *IACR Transactions on Symmetric Cryptology*, 2020(1):121–143, 2020. doi:10.13154/tosc.v2020.i1.121-143.
52. Bertram Poettering, Paul Rösler, Jörg Schwenk, and Douglas Stebila. SoK: Game-based security models for group key exchange. In Kenneth G. Paterson, editor, *Topics in Cryptology – CT-RSA 2021*, volume 12704 of *Lecture Notes in Computer Science*, pages 148–176. Springer, Heidelberg, May 2021. doi:10.1007/978-3-030-75539-3_7.
53. Ronald L. Rivest, Adi Shamir, and Leonard M. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the Association for Computing Machinery*, 21(2):120–126, 1978.
54. Phillip Rogaway. Authenticated-encryption with associated-data. In Vijayalakshmi Atluri, editor, *ACM CCS 2002: 9th Conference on Computer and Communications Security*, pages 98–107. ACM Press, November 2002. doi:10.1145/586110.586125.

55. Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Cliptography: Clipping the power of kleptographic attacks. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology – ASIACRYPT 2016, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 34–64. Springer, Heidelberg, December 2016. doi:10.1007/978-3-662-53890-6_2.
56. Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Destroying steganography via amalgamation: Kleptographically CPA secure public key encryption. Cryptology ePrint Archive, Report 2016/530, 2016. https://eprint.iacr.org/2016/530.
57. Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Generic semantic security against a kleptographic adversary. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *ACM CCS 2017: 24th Conference on Computer and Communications Security*, pages 907–922. ACM Press, October / November 2017. doi:10.1145/3133956.3133993.
58. Alexander Russell, Qiang Tang, Moti Yung, and Hong-Sheng Zhou. Correcting subverted random oracles. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 241–271. Springer, Heidelberg, August 2018. doi:10.1007/978-3-319-96881-0_9.
59. Bruce Schneier, Matthew Fredrikson, Tadayoshi Kohno, and Thomas Ristenpart. Surreptitiously weakening cryptographic systems. Cryptology ePrint Archive, Report 2015/097, 2015. https://eprint.iacr.org/2015/097.
60. Gustavus J. Simmons. The prisoners' problem and the subliminal channel. In David Chaum, editor, *Advances in Cryptology – CRYPTO'83*, pages 51–67. Plenum Press, New York, USA, 1983.
61. Yi Wang, Rongmao Chen, Xinyi Huang, and Baosheng Wang. Secure anonymous communication on corrupted machines with reverse firewalls. *IEEE Transactions on Dependable and Secure Computing*, pages 1–1, 2021. doi:10.1109/TDSC.2021.3107463.
62. Adam Young and Moti Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO'96*, volume 1109 of *Lecture Notes in Computer Science*, pages 89–103. Springer, Heidelberg, August 1996. doi:10.1007/3-540-68697-5_8.
63. Adam Young and Moti Yung. Kleptography: Using cryptography against cryptography. In Walter Fumy, editor, *Advances in Cryptology – EUROCRYPT'97*, volume 1233 of *Lecture Notes in Computer Science*, pages 62–74. Springer, Heidelberg, May 1997. doi:10.1007/3-540-69053-0_6.

Game $\mathsf{IND\text{-}CCA}^b(\mathcal{A})$
00 $C \leftarrow \emptyset$
01 $(pk, sk) \leftarrow \mathsf{KEM.gen}$
02 $b' \leftarrow \mathcal{A}^{\mathrm{Encap,Decap}}(pk)$
03 stop with b'

Oracle Encap
04 $(k^0, c) \leftarrow \mathsf{KEM.enc}(pk)$
05 $k^1 \leftarrow_{\$} \mathcal{K}$
06 $C \xleftarrow{\cup} \{c\}$
07 return (k^b, c)

Oracle Decap(c)
08 require $c \notin C$
09 $k \leftarrow \mathsf{KEM.dec}(sk, c)$
10 return k

Game $\mathsf{subIND\text{-}CCA}^b(\mathcal{A})$
00 $C \leftarrow \emptyset$
01 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
02 $(pk, sk) \leftarrow \mathsf{KEM.gen}_{i_{\mathsf{gen}}}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Encap,Decap}}(pk)$
04 stop with b'

Oracle Encap
05 $(k^0, c) \leftarrow \mathsf{KEM.enc}_{i_{\mathsf{S}}}(pk)$
06 $k^1 \leftarrow_{\$} \mathcal{K}$
07 $C \xleftarrow{\cup} \{c\}$
08 return (k^b, c)

Oracle Decap(c)
09 require $c \notin C$
10 $k \leftarrow \mathsf{KEM.dec}_{i_{\mathsf{R}}}(sk, c)$
11 return k

Fig. 10 Games modelling indistinguishability under chosen-ciphertext attacks (IND-CCA), and subverted indistinguishability under chosen-ciphertext attacks (subIND-CCA) for a key encapsulation mechanism KEM.

Game $\mathsf{IND\text{-}CCA}^b(\mathcal{A})$
00 $C \leftarrow \emptyset$
01 $k \leftarrow_{\$} \mathsf{DEM.gen}$
02 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
03 stop with b'

Oracle $\mathrm{Enc}(m^0, m^1)$
04 require $C = \emptyset$
05 $c \leftarrow \mathsf{DEM.enc}(k, m^b)$
06 $C \xleftarrow{\cup} \{c\}$
07 return c

Oracle $\mathrm{Dec}(c)$
08 require $C \neq \emptyset$
09 require $c \notin C$
10 $m \leftarrow \mathsf{DEM.dec}(k, c)$
11 return m

Game $\mathsf{subIND\text{-}CCA}^b(\mathcal{A})$
00 $C \leftarrow \emptyset$
01 $i_{\mathsf{gen}}, i_{\mathsf{S}}, i_{\mathsf{R}} \leftarrow_{\$} \mathcal{I}_{\mathsf{gen}} \times \mathcal{I}_{\mathsf{S}} \times \mathcal{I}_{\mathsf{R}}$
02 $k \leftarrow_{\$} \mathsf{DEM.gen}_{i_{\mathsf{gen}}}$
03 $b' \leftarrow \mathcal{A}^{\mathrm{Enc,Dec}}$
04 stop with b'

Oracle $\mathrm{Enc}(m^0, m^1)$
05 require $C = \emptyset$
06 $c \leftarrow \mathsf{DEM.enc}_{i_{\mathsf{S}}}(k, m^b)$
07 $C \xleftarrow{\cup} \{c\}$
08 return c

Oracle $\mathrm{Dec}(c)$
09 require $C \neq \emptyset$
10 require $c \notin C$
11 $m \leftarrow \mathsf{DEM.dec}_{i_{\mathsf{R}}}(k, c)$
12 return m

Fig. 11 Games modelling indistinguishability under one-time chosen-ciphertext attacks (IND-CCA), and subverted indistinguishability under one-time chosen-ciphertext attacks (subIND-CCA) for a data encapsulation mechanism DEM.

A Key and Data Encapsulation Mechanisms

A.1 Key Encapsulation Mechanisms

For completeness, we give the corresponding definitions of subversion attacks against key encapsulation mechanisms, together with notions of undetectability and key recovery.

A.1.1 KEM Definition

A KEM scheme $\mathsf{KEM} = (\mathsf{KEM.gen}, \mathsf{KEM.enc}, \mathsf{KEM.dec})$ for a finite session key space $\mathcal{K}$ is a triple of algorithms together with a key space $\mathcal{K}_{\mathsf{S}} \times \mathcal{K}_{\mathsf{R}}$ and ciphertext space $\mathcal{C}$. The key generation algorithm $\mathsf{KEM.gen}$ returns a pair $(pk, sk) \in \mathcal{K}_{\mathsf{S}} \times \mathcal{K}_{\mathsf{R}}$ consisting of a public key and a secret key. The encapsulation algorithm $\mathsf{KEM.enc}$ takes a public key pk to produce a session key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$. Finally, the decapsulation algorithm $\mathsf{KEM.dec}$ takes a secret key sk and a ciphertext $c \in \mathcal{C}$, and outputs either a session key $K \in \mathcal{K}$ or the special symbol $\perp \notin \mathcal{K}$ to indicate rejection. The correctness requirement is that for all $(pk, sk) \in \mathcal{K}_{\mathsf{S}} \times \mathcal{K}_{\mathsf{R}}$ we have $\Pr[\mathsf{KEM.dec}(sk, c) \neq k] \leq \delta$ for $(k, c) \leftarrow \mathsf{KEM.enc}(pk)$.

A.1.2 IND-CCA

For a key encapsulation mechanism, we formalise the indistinguishability under chosen-ciphertext attack via the game IND-CCA in Fig. 10 (left). For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}(\mathcal{A}) := \left|\Pr\left[\mathsf{IND\text{-}CCA}^0(\mathcal{A})\right] - \Pr\left[\mathsf{IND\text{-}CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme KEM is indistinguishable against chosen-ciphertext attacks if $\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{KEM}}(\mathcal{A})$ is negligibly small for all realistic $\mathcal{A}$.

A.1.3 Subverting KEM

We note that KEM schemes satisfy the generic syntax introduced above in Section 4, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{KEM.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{KEM.enc}$, receiver algorithm $\Pi.\mathsf{R} = \mathsf{KEM.dec}$. We may thus apply the generic notions of subversion introduced in Section 4.1, and observe that the passive attack in Section 8.2 applies. If the KEM scheme is in addition ciphertext sparse, according to the notion in Section 8.3.1, then the attacks in Section 8.3 will also apply. Figure 10 (right) shows the game modelling subverted indistinguishability under chosen-ciphertext attacks.

A.2 Data Encapsulation

A DEM scheme $\mathsf{DEM} = (\mathsf{DEM.gen}, \mathsf{DEM.enc}, \mathsf{DEM.dec})$ is a triple of algorithms together with associated key space $\mathcal{K}$, message space $\mathcal{M}$ and ciphertext space $\mathcal{C}$. The key generation algorithm $\mathsf{DEM.gen}$ returns key $k \in \mathcal{K}$. The encapsulation algorithm $\mathsf{DEM.enc}$ takes key $k \in \mathcal{K}$ and a message $m \in \mathcal{M}$, and outputs a ciphertext $c \in \mathcal{C}$. The decapsulation algorithm $\mathsf{DEM.dec}$ takes a key $k \in \mathcal{K}$ and a ciphertext $c \in \mathcal{C}$, and outputs either a message $m \in \mathcal{M}$ or the special symbol $\perp \notin \mathcal{M}$ to indicate rejection. The correctness requirement is that for all keys $k \in \mathcal{K}, m \in \mathcal{M}$ it holds that $\Pr[\mathsf{DEM.dec}(k,c) \neq m] \leq \delta$ for $c \leftarrow \mathsf{DEM.enc}(k,m)$.

A.2.1 IND-CCA

We formalise the indistinguishability under *one-time* chosen-ciphertext attack of a data encapsulation mechanism via the game IND-CCA in Fig. 11 (left). Note how lines 04 and 08 ensure that the adversary's first query is an encryption query, and that all further queries are decryption queries. (This precisely matches the typical situation as it emerges in a KEM/DEM hybrid.) For any adversary $\mathcal{A}$ we define the advantage

$$\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathsf{DEM}}(\mathcal{A}) := \left|\Pr\left[\mathsf{IND\text{-}CCA}^0(\mathcal{A})\right] - \Pr\left[\mathsf{IND\text{-}CCA}^1(\mathcal{A})\right]\right|$$

and say that scheme DEM is indistinguishable against chosen-ciphertext attacks if $\mathsf{Adv}^{\mathsf{ind\text{-}cca}}_{\mathcal{A}}$ is negligibly small for all realistic $\mathcal{A}$.

A.2.2 Subversion of DEM

We note that Data Encapsulation Mechanism schemes satisfy the generic syntax introduced above in Section 4, with key generation algorithm $\Pi.\mathsf{gen} = \mathsf{DEM.gen}$, sender algorithm $\Pi.\mathsf{S} = \mathsf{DEM.enc}$, receiver algorithm $\Pi.\mathsf{R} = \mathsf{DEM.dec}$. We may thus apply the generic notions of subversion introduced in Section 4.1, and observe that the passive attack in Section 8.2 applies. If the DEM scheme is in addition ciphertext sparse, according to the notion in Section 8.3.1, then the attacks in Section 8.3 will also apply. Figure 11 (right) shows the game modelling subverted indistinguishability under chosen-ciphertext attacks.

DISCUSSION. Typically, a DEM is used together with a KEM in a so-called hybrid encryption scheme that uses the KEM to share (symmetric) session keys with which plaintext messages are encrypted under the DEM. In such a setting, subverting the KEM is sufficient to undermine the security of messages sent via the hybrid scheme. Following the discussion at Section 4.3, it is conceivable to subvert a KEM and DEM in tandem so that the KEM's secret key is leaked by the both together. This could allow the subversion to effectively be distributed between the two primitives, aiding undetectability in practice.

Proc $\mathsf{CS.gen}(\mathbb{G}, g, \hat{g}, \mathsf{hk})$
00 $x_1, x_2, y_1, y_2, z_1, z_2 \leftarrow_{\$} \mathbb{Z}_q$
01 $a \leftarrow g^{x_1}\hat{g}^{x_2}, b \leftarrow g^{y_1}\hat{g}^{y_2}, d \leftarrow g^{z_1}\hat{g}^{z_2}$
02 $pk \leftarrow (a, b, d)$
03 $sk \leftarrow (x_1, x_2, y_1, y_2, z_1, z_2)$
04 output (sk, pk)

Proc $\mathsf{CS.enc}(pk, m)$
05 $u \leftarrow_{\$} \mathbb{Z}_q, w \leftarrow g^u, \hat{w} \leftarrow \hat{g}^u$
06 $e \leftarrow d^u \cdot m$
07 $\rho \leftarrow H_{\mathsf{hk}}(a, \hat{a}, e)$
08 $v \leftarrow a^u b^{u\rho}$
09 output $c = (a, \hat{a}, e, v)$

Proc $\mathsf{CS.dec}(sk, c)$
10 parse c as $(a, \hat{a}, e, v)$
11 $\rho \leftarrow H_{\mathsf{hk}}(a, \hat{a}, e)$
12 if $v \neq a^{x_1+y_1\rho} \cdot \hat{a}^{x_2+y_2\rho}$
13 return $\perp$
14 else:
15 $m \leftarrow c \cdot (a^{z_1}\hat{a}^{z_2})^{-1}$
16 return m

Fig. 12 Cramer-Shoup PKE scheme $\mathsf{CS} = (\mathsf{CS.gen}, \mathsf{CS.enc}, \mathsf{CS.dec})$.

B Example Ciphertext Sparse PKE Schemes

We describe two widespread PKE schemes that satisfy the notion of ciphertext sparseness described in Section 8.3.1.

OAEP. Optimal Asymmetric Encryption Padding (OAEP) was introduced by Bellare and Rogaway [15] and is a widely deployed and standardised PKE scheme. The encryption algorithm of OAEP works on message space $\mathcal{M} = \{0,1\}^\ell$ with fixed message length ℓ. Let k_0 and k_1 be integers, and $G\colon \{0,1\}^{k_0} \to \{0,1\}^{\ell+k_1}$ and $H\colon \{0,1\}^{\ell+k_1} \to \{0,1\}^{k_0}$ be two hash functions. Messages are padded before being encrypted using the trapdoor permutation (typically RSA): To pad a message $m \in \mathcal{M}$, set $m' \leftarrow m \,\|\, 0^{k_1}$ and choose $r \leftarrow_{\$} \{0,1\}^{k_0}$. Then set $s \leftarrow m' \oplus G(r)$, $t \leftarrow r \oplus H(s)$ and $\hat{m} \leftarrow s \,\|\, t$. To decrypt a ciphertext, first decrypt (i.e. apply the trapdoor inverse) before unpadding the resulting padded message $\hat{m}$: Parse $\hat{m}$ as $s \,\|\, t$ with $s \in \{0,1\}^{\ell+k_1}$ and $t \in \{0,1\}^{k_0}$. Now compute $r \leftarrow t \oplus H(s)$ and $m' \leftarrow s \oplus G(r)$. If $m' \neq m \,\|\, 0^{k_1}$ for some m then reject, otherwise return m.

For a randomly chosen element c in the ciphertext space $\mathcal{C}$, the redundancy introduced by padding will ensure that decrypting c results in a valid message with probability 2^{-k_1}. This is because choosing $c \leftarrow_{\$} \mathcal{C}$ is equivalent to choosing a random $m' \leftarrow_{\$} \{0,1\}^{\ell+k_1}$, assuming that the trapdoor permutation and hash functions all behave like random functions. Equivalently, the scheme is ciphertext 2^{-k_1}-sparse, according to the definition in Section 8.3.1.

CRAMER-SHOUP. The Cramer-Shoup PKE scheme was introduced in [26]. The encryption scheme $\mathsf{CS} = (\mathsf{CS.gen}, \mathsf{CS.enc}, \mathsf{CS.dec})$ is defined in relation to a set of public parameters consisting of finite group $\mathbb{G}$ with $|\mathbb{G}| = q$ and a pair of generators $g, \hat{g}$ for $\mathbb{G}$, together with a hash key hk for a family of keyed collision resistant universal hash functions $H_{\mathsf{hk}} : \mathbb{G}^3 \to \mathbb{Z}_q$. The family of keyed hash functions is such that given a randomly chosen tuple of group elements and randomly chosen hash function key, it is computationally infeasible to find a different tuple of group elements that hashes to the same value using the given hash key. We give details of Cramer-Shoup in Fig. 12.

For a randomly chosen element $c = (a, \hat{a}, e, v)$ in the ciphertext space $\mathbb{G}^4$, the redundancy introduced by the hash function will ensure that decrypting c results in a valid message with probability q^{-1}. To see this, consider fixed $a, \hat{a}, e$: this gives a fixed value of $a^{x_1+y_1\rho} \cdot \hat{a}^{x_2+y_2\rho} \in \mathbb{G}$ and thus $\Pr[v = a^{x_1+y_1\rho} \cdot \hat{a}^{x_2+y_2\rho}] = q^{-1}$.

C Plaintext Awareness

We here give the definition of plaintext awareness, a property of PKE schemes that implies ciphertext sparseness (see Section 8.3.1). Plaintext awareness essentially means that an adversary is unable to create ciphertexts without knowing the underlying plaintext message. This means that ciphertexts which have not been generated from underlying

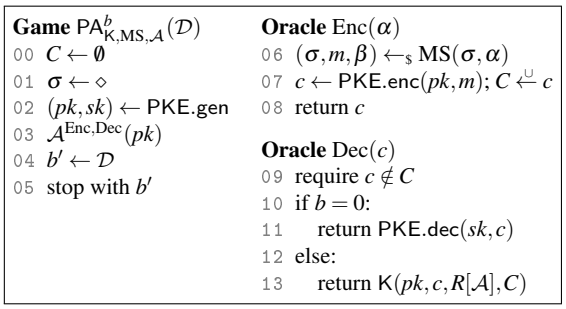

Fig. 13 Game modelling plaintext awareness (PA), for a public key encryption scheme PKE. Note that we retain β (modelling side-channel information) in the syntax of message sampler MS for consistency with previous sections, but here the adversary is not given output β.

plaintext messages should be rejected, implying that ciphertexts chosen uniformly at random from the ciphertext space are unlikely to be valid. Both the Cramer-Shoup cryptosystem [25,31] and RSA+OAEP [53,15], outlined in Section 8.3.1, satisfy plaintext awareness.

The formal definitions of plaintext awareness in the standard model were proposed by Bellare and Palacio [13], and were slightly extended by Dent and Birkett [31,20]. A scheme is plaintext aware if for all ciphertext creators (attackers) $\mathcal{A}$, there exists a plaintext extractor K which takes as input the random coins of A and can answer the decryption queries of A in a manner that A cannot distinguish from a real decryption oracle. In order to model the attacker's ability to obtain ciphertexts for which it does not know the underlying plaintext, the ciphertext creator is equipped with an oracle that will return the encryption of a randomly chosen message $m \leftarrow_{\$} \mathrm{MS}$, where MS is an arbitrary (stateful) message sampling algorithm that takes as input α allowing an adversary (ciphertext creator) to specify a distribution on messages. In Fig. 13, we write $\diamond$ for the initial state of MS. After interacting with either the real decryption algorithm or the knowledge extractor simulating decryption, the ciphertext creator outputs a ciphertext c. A distinguisher $\mathcal{D}$ is now tasked with guessing which case we are in. Note that the knowledge extractor K does not have access to the distinguisher's randomness.

We formalise plaintext awareness of a public key encryption scheme via the game PA in Fig. 13. For any distinguisher $\mathcal{D}$ we define the advantage

$$\mathsf{Adv}^{\mathsf{pa}}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D}) := \left|\Pr\left[\mathsf{PA}^{0}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})\right] - \Pr\left[\mathsf{PA}^{1}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})\right]\right|.$$

We say that a scheme is plaintext aware if for all realistic ciphertext creators $\mathcal{A}$, there exists a knowledge extractor K such that for all message samplers MS and distinguishers $\mathcal{D}$, the advantage $\mathsf{Adv}^{\mathsf{pa}}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})$ is negligibly small.

We note that a PKE scheme that satisfies plaintext awareness and indistinguishability against chosen ciphertext attacks (Section 7) is necessarily ciphertext sparse. To see this, suppose that the PKE scheme is not ciphertext sparse. For a randomly chosen ciphertext $c \leftarrow_{\$} \mathcal{C}$, the real game $\mathsf{PA}^{0}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})$ will output a valid message m. However, in the random game $\mathsf{PA}^{1}_{\mathsf{K},\mathrm{MS},\mathcal{A}}(\mathcal{D})$ the knowledge extractor will not be able to output m without contradicting the plaintext awareness and CCA security of the scheme. We thus conclude that the scheme is ciphertext sparse.