Algorithm Substitution Attacks on Public Functions

Mihir Bellare[1] Doreen Riepel[2] Laura Shea[3]

June 14, 2024

Abstract

We study the possibility of algorithm substitution attacks (ASAs) on functions with no secret-key material, such as hash functions, and verification algorithms of signature schemes and proof systems. We consider big-brother's goal to be three-fold: It desires utility (it can break the scheme), exclusivity (nobody else can) and undetectability (outsiders can't detect its presence). We start with a general setting in which big-brother is aiming to subvert an arbitrary public function. We give, in this setting, strong definitions for the three goals. We then present a general construction of an ASA, and prove that it meets these definitions. We use this to derive, as applications, ASAs on hash functions, signature schemes and proof systems. As a further application of the first two, we give an ASA on X.509 certificates. While ASAs were traditionally confined to exfiltrating secret keys, our work shows that they are possible and effective at subverting public functions where there are no keys to exfiltrate. Our constructions serve to help defenders and developers identify potential attacks by illustrating how they might be built.

[1] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: mbellare@ucsd.edu. URL: http://cseweb.ucsd.edu/~mihir/. Supported in part by NSF grant CNS-2154272 and KACST.

[2] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: driepel@ucsd.edu. Supported in part by KACST.

[3] Department of Computer Science & Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, California 92093, USA. Email: lmshea@ucsd.edu. Supported by NSF grants CNS-2048563, CNS-1513671 and CNS-2154272.

Contents

1 Introduction

The Snowden revelations lead researchers to ask how cryptography might be subverted. Algorithm Substitution Attacks (ASAs) [10, 66, 67] emerged as one answer. The first formalism and attacks, by BPR [10], put forth the following template. There is a prescribed cryptographic algorithm A that accesses a secret key K. The adversary (called big-brother in this setting) substitutes A by subverted code $\tilde{\mathsf{A}}$ that aims to undetectably exfiltrate K. For this purpose, $\tilde{\mathsf{A}}$ and big-brother may share a symmetric key. Such ASAs have now been given or considered for many primitives including symmetric encryption [3, 9, 10, 26, 41], signature schemes [5, 20, 62] and beyond [14, 14, 19, 21, 38, 44, 53–55, 63]. However, the goal has always remained to exfiltrate a *secret key*.

We initiate work in a new and different direction inspired by work on backdooring of machine-learning algorithms [40]. We consider the possibility of ASAs on public functions, such as hash functions, verification algorithms of signature schemes, or verification algorithms of proof systems like SNARGs or SNARKs. In these cases, there is no secret key to exfiltrate, so one has to ask what an ASA would want to do, and what properties big-brother would like it to have. We answer these questions with new definitions. These are given in a general setting where the goal is an ASA on an arbitrary public function. In the same general setting, we then give a construction of an ASA that we prove meets our definitions. We then apply this to obtain ASAs on the specific primitives mentioned above.

The setting. Honest function f is sampled from a prescribed family F, as $f \leftarrow\!\!\$\ \mathsf{F}$. Big-brother generates a substitution function $\widetilde{f}$ together with an associated exploit function (also called a backdoor) e, via $(\widetilde{f}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{F}}(f)$, where $\widetilde{\mathsf{F}}$ is an algorithm of big-brother's own devising. It now arranges that a user's code implementing f is substituted with code implementing $\widetilde{f}$. This substitution can take place by a variety of means and is outside our scope. Applications that expected to use f are now (unknowingly) using $\widetilde{f}$ instead. In this setting, we consider big-brother to have three goals, as follows:

- Utility: Big-brother, through knowledge of the exploit function e, now wants to violate security of the $\widetilde{f}$-using application.
- Undetectability: A tester with black-box access to either f or $\widetilde{f}$ should not be able to tell which of the two it is.
- Exclusivity: Others, meaning entities knowing $\widetilde{f}$ but not e, should *not* be able to violate security of the application even when it uses $\widetilde{f}$.

Utility of course represents the main intent of big-brother in creating the ASA. Undetectability is a core ASA requirement [10]. Exclusivity captures that a big-brother government or Intelligence Agency wants to ensure that other parties cannot violate security of the application. Similarly, a big-brother corporation may want to ensure that its competitors and the public cannot violate security of the application.

Motivating applications. Before formalizing the above requirements and giving our general construction, we discuss some motivating applications.

The first is that f is a collision-resistant hash function. The canonical application is hashing the content (data) in a digital certificate before signing it to create the certificate itself. The utility of interest to big-brother would be that, when $\widetilde{f}$ is used in place of f, big-brother can forge certificates. Exclusivity means others cannot forge certificates even under $\widetilde{f}$. The ASAs we will give improve the backdoored hash functions of [34] which had lower utility, lower exclusivity and no undetectability requirement.

The second is signatures. Prior work on ASAs [5] aims to substitute the signing algorithm and

exfiltrate the secret signing key, and the ASAs only succeed for schemes which have randomized signing with sufficient entropy. Instead, we let f be the verification algorithm, viewed as embedding a target, public verification key vk. The substitution continues to accept signatures under vk, but utility allows big-brother to have it also accept other signatures that big-brother devises using e. Yet, exclusivity ensures that an outside entity not knowing either e or the secret signing key underlying vk be unable to create new signatures that $\widetilde{f}$ accepts.

The third is proof systems. These are extensively used in blockchains and cryptocurrencies. Our ASAs will show how to subvert soundness. We let f be the proof-verification algorithm, possibly embedding a CRS. The substitution $\widetilde{f}$ accepts valid proofs under the CRS but also accepts proofs, devised by big-brother using e, of false statements. Exclusivity precludes others from violating soundness even relative to $\widetilde{f}$.

The general framework. Rather than treat the above three applications separately, we will derive them via a general framework that we now discuss. Here F is an arbitrary function family from which the honest function $f \leftarrow\!\!\$\ \mathsf{F}$ is sampled, and big-brother generates the substitution function $\widetilde{f}$ together with an exploit function e via $(\widetilde{f}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{F}}(f)$.

- Utility: Utility seems at first glance to be very application dependent, and we have to ask what it may mean in general. We parameterize the requirement by a predicate $\mathsf{P}(\cdot,\cdot)$ called the constraint. Then we ask that exploit function e, given a string u called the constraint-parameter, and given a target output y, returns an x such that (1) $\widetilde{f}(x) = y$ and (2) $\mathsf{P}(x, u) = \mathsf{true}$. Intuitively, big-brother can create an input x, satisfying a desired constraint involving a u of its choice, such that x maps to an output y of its choice under $\widetilde{f}$. Different applications will make different choices of P.
- Undetectability: We consider a game that picks a random challenge bit b and gives the detector an oracle that on input x returns $f(x)$ if $b = 1$ and $\widetilde{f}(x)$ otherwise. Asking that the detector have negligible advantage in guessing b formalizes black-box undetectability, a strong requirement in line with prior work [10, 40].
- Exclusivity: We consider an adversary that is given the descriptions of f and $\widetilde{f}$ and oracle access to the exploit function e, and ask that it cannot come up with an input x at which f and $\widetilde{f}$ differ, except trivially, meaning through use of its oracle.

Our first result is Theorem 3.1, saying that exclusivity, in our strong formulation, actually implies undetectability. We define the latter separately because it is a core ASA requirement insisted on by prior work [10], but for our ASAs, we will prove exclusivity and then conclude undetectability via Theorem 3.1.

Our general **ASA** construction. Is it possible to build an ASA $\widetilde{\mathsf{F}}$ on an arbitrary function family F that meets the three conditions above? We show, through construction, that the answer is "yes."

To expand on this, first note that one cannot hope to achieve this for all constraint predicates P. It is, for example, impossible for the predicate $\mathsf{P}(x, u)$ that returns true iff $x = u$; intuitively, one needs some "room" in x to exploit. We show how to build an ASA $\widetilde{\mathsf{F}}$ for any constraint predicate satisfying a certain condition, that we define and call *embeddability*. The class of predicates meeting this condition is large and includes in particular ones allowing our above-discussed applications.

Our construction is a transform **ASA** that takes (1) the target function family F (2) a signature scheme S and (3) an *embedding function* Emb, for the constraint predicate P, that is compatible with S, F, as we will define in Section 4. (Embeddability, for now, just asks that such an embedding exists.) It returns an ASA $\widetilde{\mathsf{F}} = \mathbf{ASA}[\mathsf{F}, \mathsf{S}, \mathsf{Emb}]$ built from these three components. Proposition 4.1 establishes utility with respect to P, assuming correctness of the embedding and signature scheme.

Theorem 4.2 establishes exclusivity assuming strong unforgeability of the signature scheme S. Undetectability follows from Theorem 3.1. The design of **ASA** extends ideas of [40], who used a signature scheme to backdoor a machine learning model.

ASAs on hash functions. Towards our first application, to collision-resistant hash functions and certificate forgeries, we start with some background. While ASAs on hash functions have not to our knowledge been explicitly discussed before, Fischlin, Janson and Mazaheri (FJM) [34] considered backdooring hash functions, which is similar. Let H be the target set of hash functions from which, via $h \leftarrow\!\!\$\ \mathsf{H}$, one generates an honest hash function $h : \{0,1\}^* \to \{0,1\}^\ell$ that is assumed collision-resistant. A backdooring of H can be seen, like an ASA, as specified by an algorithm $\widetilde{\mathsf{H}}$ that takes an instance h of H and via $(\tilde{h}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{H}}(h)$ generates a substitution function $\tilde{h} : \{0,1\}^* \to \{0,1\}^\ell$ as well as a backdoor (or exploitation) function e. FJM [34] have two requirements that in our language represent utility and exlusivity. FJM-utility asks that knowledge of e allows big-brother to violate collision resistance (cr) of $\tilde{h}$, meaning find distinct x_1, x_2 such that $\tilde{h}(x_1) = \tilde{h}(x_2)$. FJM-exclusivity asks that an adversary given $\tilde{h}$ but not e cannot violate cr of $\tilde{h}$. They give no undetectability condition.

FJM [34] build a backdooring $\widetilde{\mathsf{H}}$ of H satisfying their two conditions. Roughly, $\tilde{h}$ embeds the image $\overline{y}$ of a random point $\overline{x}$ under a one-way function g, and $\tilde{h}$ behaves anomalously if (and only if) its input maps to $\overline{y}$ under g. Other works also satisfy this basic notion of a backdoored hash function: Albertini, Aumasson, Eichlseder, Mendel and Schläffer (AAEMS) [2] give a backdoored version of SHA1. The VSH (Very Smooth Hash) algorithm of Contini, Lenstra and Steinfeld [22] achieves FJM-utility and FJM-exclusivity when viewed as a backdoored hash function.

This notion of utility is however limited; in its quest for subversion, big-brother wants significantly more. We illustrate with two examples:

Forgery of TLS certificates. Forgery of TLS certificates, to allow impersonation of a target website, is a commonly considered big-brother goal [60, 61, 69]. Hashing is used here by the CA to compress the certificate data before it signs to create the certificate. An arbitrary hash collision will not help towards a forgery; big-brother needs not only to find a preimage x of a target point y (the hash of a legitimate certificate) but to be able to embed in x information of its choice, such as a public key for which it knows the secret key.

Breaking password-based authentication. Big-brother may be interested in subverting password-based authentication, where the server holds a hash of the user password and salt. Big-brother needs to find a preimage of this same hash that contains a password that it knows and the given salt as a suffix.

We note that FJM [34] and AAEMS [2] suggest that utility be the ability to violate preimage resistance but their formulations still do not suffice for the above tasks because they do not allow big-brother to embed information of its choice in the preimage. Meanwhile, FJM-exclusivity too is limited, as collisions created by big-brother using e can reveal e, making it easy for outside observers to find further collisions.

In Section 5, we give an ASA on hash functions that satisfies our strong utility and exclusivity requirements. In particular, the ASA permits the two exploits discussed above, meaning TLS certificate forgery and breaking of password-based authentication. We obtain our hash-function ASA by applying our above-discussed general **ASA** and results about it. To obtain the two exploits, we make particular choices of the constraint predicates that parameterize our definition. Section 8 describes the X.509 forgery in detail. We note that this application relies crucially on our strong, predicate-parameterized definition of utility.

ASAs on proof systems. As a second example, we illustrate ASAs on proof systems, targeting

the substitution of the verification algorithm. We use the notation $(p, v) \leftarrow\!\!\$\ \mathsf{PS}$ to denote the generation of a proving and verification algorithm which hardcode the CRS. Our ASA then targets the family of verification functions V, where our goal is to replace $v \in \mathsf{V}$ with a substitution function $\widetilde{v}$, and to produce an exploit function e via an ASA generation algorithm $\widetilde{\mathsf{V}}$.

The property we presume an ASA on verification to target is soundness. Using e, we want to be able to create an accepting proof for any statement ϕ which is not in the language described by the proof system. This means that our utility definition asks to effectively violate soundness. Further, we ask that soundness is maintained for any other party not having access to e, even if it has observed forged proofs for statements of its choice. This is captured by our exclusivity definition which provides oracle access to e. In order to apply our general **ASA** transform, we view a proof forgery as finding a preimage of 1. (Recall that a verification function outputs a bit where 1 indicates success.) In line with the general setting, we define a corresponding predicate P_{PS} along with an embedding function. We include these results in Section 6.

ASAs on signature schemes. Finally, we turn to signature schemes. Similar to proof systems, we want to substitute the verification algorithm. However in this setting, different users create their own signing-verification key pair, hence it is desired that an exploitation algorithm works for not only one but multiple users. Again we consider a family of functions V, where a function $v_i \in \mathsf{V}$ hardcodes the verification key for user i, and we want to construct an ASA $\widetilde{\mathsf{V}}$ which takes $v_1, \ldots, v_n$ and generates substitution functions $\widetilde{v}_1, \ldots, \widetilde{v}_n$ and a single exploit function e for all n users.

Utility might ask that possession of e allows creation of some forgery, meaning a message m and a valid signature for it under the verification function $\widetilde{v}_i$ of some user. We ask for a stronger notion of utility, namely that given any user i, and *any target message* m, the exploit algorithm allows creation of a signature for m relative to $\widetilde{v}_i$. Our exclusivity definition is similarly strong, asking that unforgeability of signatures still holds even with access to an oracle for producing signatures using the exploit algorithm for any user. We observe that our general ASA transform needs to be adapted so that signatures produced by the exploit algorithm are tied to the user. In Section 7, we therefore give a modified multi-user ASA transform $\mathbf{ASA}_n$ and show that it indeed achieves our specified goals above.

As a concrete application, we consider again the certificate forgery example. Using our signature ASA, big-brother can use the exploitation function to create certificates with validating signatures. This is discussed in detail in Section 8.

Related work. "Subversion" is a recurring concern in cryptography. Attacks have been observed in a variety of settings, including parameter generation in Dual EC [15, 57], malicous code changes in Linux [32], and governmental exceptional access [1]. To situate our investigation of ASAs on public functions, it is useful to consider different categories in this area of subverted cryptography.

In a first category, code can be maliciously modified from its algorithmic specification. This has been studied as algorithm substitution attacks [10] and as kleptography [66–68]. Here, an adversary's goal is to both modify an algorithm such that it exfiltrates secret information, and to keep this modification undetected. We call our considered attacks ASAs because they involve a code substitution step, and a second exploitation step. However, we define different goals than exfiltration and undetectability. ASAs and kleptography have been studied for symmetric encryption [3, 9, 10, 26], KEMs [19, 44, 53], signatures [5, 62], and protocols [14, 14, 21, 38, 63]. Defenses have been studied from the perspective of preventing exfiltration [30, 50] or other "subversion-resistant" notions [4, 13, 54, 55]. These include more fine-grained online/offline detectability notions and other modes of computation.

In another category of subversion, violations occur through authoritarian means. That is, an authority can overcome usual security guarantees, but this extra power is well known to the

public. Anamorphic cryptography [7, 47, 52] and earlier work on subliminal channels [43, 58, 59] have proposed defenses to this type of subversion, which is different than the ASA model.

A final category, somewhat more related to ASAs, is maliciously designed algorithms or parameters. These have been studied for PRGs [27, 29], NIZKs [8], PKE [6], and hash functions [2, 34]. Unlike an ASA, algorithms are assumed to be implemented honestly, and code can be inspected. Nonetheless, some of the goals and techniques are similar. In relevant sections, we will compare related work on particular primitives in more detail.

The 2022 work of Goldwasser, Kim, Vaikuntanathan and Zamir (GKVZ) studies the possibility of inserting undetectable backdoors in a machine learning model [40]. We leverage their techniques in Section 4. In GKVZ, a strongly unforgeable signature triggers alternate execution in a model, modifying classifier output when a signature is correctly parsed and verified. The non-replicability condition of GKVZ offers an oracle providing backdoored model inputs, which is similar to an additional oracle that our exclusivity game provides. We note that the predicates and embeddings we additionally formalize in Section 4 generalize the parsing of GKVZ, and that our techniques for "general public functions" seem to be quite applicable beyond cryptographic functions.

As a real-world motivation to study ASAs, the `xz` backdoor was discovered on March 29, 2024 [35]. Current understanding of its cryptographic portion [64] shows interesting similarities with our ASAs, such as the embedding of a signature and attacker-chosen data in a certificate which triggers alternate execution during certificate validation. The discovery of the `xz` backdoor shows that ASAs targeting high levels of utility are a realistic possibility, and motivates research, such as ours, on this topic.

2 Preliminaries

NOTATION AND TERMINOLOGY. By ε we denote the empty string. By $|Z|$ we denote the length of a string Z. By $x \,\|\, y$ we denote the concatenation of strings x, y. If Z is a string, we let $Z[a..b]$ be the substring of Z between indices a and b, inclusive, or ε if $b < a$. If S is a finite set, then $|S|$ denotes it size. We say that a set S is *length-closed* if, for any $x \in S$ it is the case that $\{0,1\}^{|x|} \subseteq S$. (This will be a requirement for certain spaces.)

If $\mathcal{X}$ is a finite set, we let $x \leftarrow\!\!\$\, \mathcal{X}$ denote picking an element of $\mathcal{X}$ uniformly at random and assigning it to x. If A is an algorithm, we let $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; r)$ denote running A on inputs $x_1, \ldots$ and coins r with oracle access to $\mathrm{O}_1, \ldots$, and assigning the output to y. We let $y \leftarrow\!\!\$\, A[\mathrm{O}_1, \ldots](x_1, \ldots)$ be the result of picking r at random and computing $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; r)$. We let $\mathrm{OUT}(A[\mathrm{O}_1, \ldots](x_1, \ldots))$ denote the set of all possible outputs of A when invoked with inputs $x_1, \ldots$ and oracles $\mathrm{O}_1, \ldots$. Algorithms are randomized unless otherwise indicated. Running time is worst case, which for an algorithm with access to oracles means across all possible replies from the oracles. The abbreviation "p.p.t." denotes "probabilistic polynomial time."

An adversary is an algorithm. We use $\perp$ (bot) as a special symbol to denote rejection, and it is assumed to not be in $\{0,1\}^*$. The image of a function $f : \mathcal{D} \to \mathcal{R}$ is the set $\mathrm{Im}(f) = \{f(x) : x \in \mathcal{D}\} \subseteq \mathcal{R}$. We may interchangeably refer to the boolean false and integer 0, or to the boolean true and integer 1.

GAMES. We use the code-based game-playing framework of BR [11]. A game G starts with an optional INIT procedure, followed by a non-negative number of additional procedures called oracles, and ends with a FIN procedure. Execution of adversary A with game G begins by running INIT (if present) to produce $\mathsf{input} \leftarrow\!\!\$\, \textsc{Init}$. A is then given input and is run with query access to the game oracles. When A terminates with some output, execution of game G ends by returning $\textsc{Fin}(\mathsf{output})$.

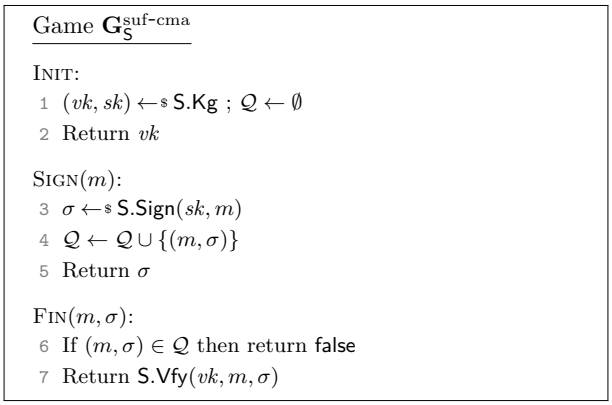

Figure 1: Strong unforgeability for a signature scheme S.

By $\Pr[\mathrm{G}(A)]$ we denote the probability that the execution of game G with adversary A results in FIN(output) being the boolean true.

Different games may have procedures (oracles) with the same names. If we need to disambiguate, we may write G.O to refer to oracle O of game G. In games, integer variables, set variables, boolean variables and string variables are assumed initialized, respectively, to 0, the empty set $\emptyset$, the boolean false and $\perp$. Tables are initialized with all entries being $\perp$. Games may occasionally Require: some condition, which means that all adversaries must obey this condition. This is used to rule out trivial wins.

UNFORGEABILITY OF SIGNATURES. A signature scheme S specifies algorithms $\mathsf{S.Kg}, \mathsf{S.Sign}, \mathsf{S.Vfy}$, key spaces $\mathsf{S.VK}, \mathsf{S.SK}$, and signature length $\mathsf{S.sl}$. Key generation $\mathsf{S.Kg}$ produces a verification key $vk \in \mathsf{S.VK}$ and signing key $sk \in \mathsf{S.SK}$ via $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$. Signing takes as input a signing key $sk \in \mathsf{S.SK}$ and message $m \in \{0,1\}^*$ to return a signature $\sigma \in \{0,1\}^{\mathsf{S.sl}}$ via $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, m)$, where $\mathsf{S.sl} \in \mathbb{N}$ is a constant signature length. Deterministic algorithm $\mathsf{S.Vfy}$ takes as input a verification key $vk \in \mathsf{S.VK}$, message $m \in \{0,1\}^*$, and signature $\sigma \in \{0,1\}^{\mathsf{S.sl}}$ to return a bit d via $d \leftarrow \mathsf{S.Vfy}(vk, m, \sigma)$.

Correctness of scheme S asks that for all $(vk, sk) \in \mathrm{OUT}(\mathsf{S.Kg})$, for all $m \in \{0,1\}^*$, it holds that $\mathsf{S.Vfy}(vk, m, \mathsf{S.Sign}(sk, m)) = 1$.

The security notion that we will make use of is strong unforgeability. This is captured by game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}$ of Figure 1. If A is an adversary, we let $\mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A) = \Pr\left[\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}(A)\right]$ be its suf-cma advantage. Strongly unforgeable signatures have been constructed based on bilinear CDH [17], strong RSA [24, 37], and generally from one-way functions [39, Section 6.5].

3 Algorithm substitution attacks on public functions

We begin with new definitions for algorithm substitution attacks on arbitrary functions. In Section 4 we will provide a construction satisfying our notions, and in the remainder of the paper we extend both our definitions and construction to more specific settings. As our first task, we introduce a generic ASA syntax.

SYNTAX. Let F be a function family; a member function $f : \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$ is selected via $f \leftarrow\!\!\$\ \mathsf{F}$. An ASA aims to substitute f with $\widetilde{f}$ which has certain malicious behavior on particular inputs, while ensuring that the ability to find such inputs is exclusive to the attacker. Concretely,

Game $\mathbf{G}^{\text{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}$	Game $\mathbf{G}^{\text{det}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}$
Init:	Init:
1 $f \leftarrow\!\!\$\ \mathsf{F}$; $(\widetilde{f}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{F}}(f)$	1 $f \leftarrow\!\!\$\ \mathsf{F}$; $(\widetilde{f}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{F}}(f)$
2 $\mathcal{X} \leftarrow \emptyset$	2 $b \leftarrow\!\!\$\ \{0,1\}$
3 Return $(f, \widetilde{f})$	3 Return ε
GetPmg(u, y):	Eval(x):
4 $x \leftarrow\!\!\$\ e(u, y)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$	4 $y_0 \leftarrow f(x)$; $y_1 \leftarrow \widetilde{f}(x)$
5 Return x	5 Return y_b
Fin(x^*):	Fin(b'):
6 Return $(x^* \notin \mathcal{X}) \wedge (f(x^*) \neq \widetilde{f}(x^*))$	6 Return $(b = b')$

Figure 2: Exclusivity (left) and detectability (right) of an ASA $\widetilde{\mathsf{F}}$ on public function f. While detectability is black-box, exclusivity returns the functions $(f, \widetilde{f})$ to the adversary.

---

we have an algorithm $\widetilde{\mathsf{F}}$ which generates $(\widetilde{f}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{F}}(f)$, where $\widetilde{f}$ is the *substitution* and e is the *exploitation* algorithm. Before elaborating on this ability, let us clarify the ASA model.

ASA model for public functions. An attacker who mounts an ASA proceeds in two steps, substitution and exploitation. First, they generate algorithms $(\widetilde{f}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{F}}(f)$ and replace a user's implementation of f with one of $\widetilde{f}$. The exploitation algorithm e remains secret and is retained by the attacker. Second, in the exploitation step, the attacker may use e to find a preimage x of target point y. The user, who has $\widetilde{f}$ on their device, now computes $\widetilde{f}(x) = y$. The interface of this exploitation algorithm is broad, applying to settings where (1) it is reasonable that the attacker chooses inputs x to send to the user, and (2) it is useful to an attacker to find preimages of target points (where the target point could even be 1 or true).

Utility. The *utility* capabilities of $\widetilde{\mathsf{F}}$ are captured with respect to a constraint predicate P. Let $\mathsf{P} : \{0,1\}^* \times \{0,1\}^* \to \{\mathsf{true}, \mathsf{false}\}$ be a predicate. We say that $\widetilde{\mathsf{F}}$ achieves utility relative to P, if for every every $f \in \mathsf{F}$, every constraint-parameter $u \in \{0,1\}^*$ and every $y \in \{0,1\}^{\mathsf{F.ol}}$, if $(\widetilde{f}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{F}}(f)$ and if $x \leftarrow\!\!\$\ e(u, y)$, then we have (1) $\widetilde{f}(x) = y$ and (2) $\mathsf{P}(x, u) = 1$. In other words, e allows one to compute a preimage of any target output, where the preimage also satisfies the constraint predicate. In the following section we discuss predicates in more detail. At a high level, this utility definition relative to a predicate allows one to specify fine-grained notions of successful attacks, beyond only finding *some* preimage.

Exclusivity. Effectiveness of an ASA calls for exclusivity as well as utility. *Exclusivity* requires that, even after seeing inputs x such that $f(x) \neq \widetilde{f}(x)$, it remains hard to find another nontrivial input x^* on which f and $\widetilde{f}$ differ. This is captured by the exclusivity game $\mathbf{G}^{\text{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}$ of Figure 2. If A is an adversary, we let $\mathbf{Adv}^{\text{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A) = \Pr\left[\mathbf{G}^{\text{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A)\right]$ be its exc advantage.

We will see exclusivity applied to more specific settings in later sections, but for now, we remark that an ASA is only useful if the substituted algorithm $\widetilde{f}$ differs from f on some inputs (else f could be attacked directly). Exclusivity asks that is hard for anyone other than the ASA to find these inputs which produce different behavior. The game $\mathbf{G}^{\text{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}$ provides an additional oracle GetPmg for seeing these "trigger inputs" and an adversary A wins if it produces another such input x^* which was not generated by GetPmg directly.

Undetectability. We define (black-box) undetectability as a distinguishing game which is de-

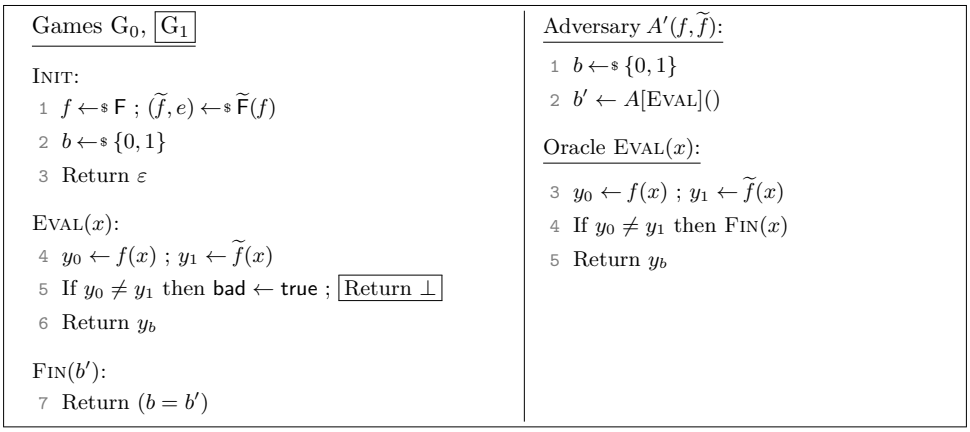

Figure 3: Games G_0, G_1 (left) and adversary A' (right) for the proof of Theorem 3.1. G_1 contains the boxed code and G_0 does not.

---

scribed on the right side of Figure 2. If A is an adversary, we let $\mathbf{Adv}^{\text{det}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A) = 2 \cdot \Pr\left[\mathbf{G}^{\text{det}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A)\right] - 1$ be its det advantage. The following statement shows that exclusivity implies undetectability. For this reason, we will continue our focus on exclusivity, as it is stronger. Nonetheless, black-box undetectability remains a standard notion in works on ASAs.

Theorem 3.1 *Let F be a family of functions and $\widetilde{\mathsf{F}}$ an ASA on F relative to a predicate P. Given an adversary A against the undetectability of $\widetilde{\mathsf{F}}$ we can build an adversary A' such that*

$$\mathbf{Adv}^{\text{det}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A) \leq 2 \cdot \mathbf{Adv}^{\text{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A') \; . \tag{1}$$

A' makes no GetPmg *queries. The running time of A' is close to that of A.*

Proof of Theorem 3.1: Consider game G_0 of Figure 3 which is exactly the det game, except that it additionally checks whether y_0 equals y_1 for queries to Eval and sets flag bad if they are not the same. The output, however, is not modified, so

$$\mathbf{Adv}^{\text{det}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A) = 2 \cdot \Pr\left[\, G_0(A) \,\right] - 1 \; .$$

We now turn to G_1. Games G_0, G_1 are identical-until-bad, so by the Fundamental Lemma of Game Playing [11] we have

$$\begin{aligned} \Pr\left[\, G_0(A) \,\right] &= \Pr\left[\, G_1(A) \,\right] + \left(\Pr\left[\, G_0(A) \,\right] - \Pr\left[\, G_1(A) \,\right]\right) \\ &\leq \Pr\left[\, G_1(A) \,\right] + \Pr\left[\, G_1(A) \text{ sets } \mathsf{bad} \,\right] . \end{aligned}$$

We construct adversary A' on the right side of Figure 3. A' gets as input functions f and $\widetilde{f}$, and can simulate the detectability game in a straightforward way by picking its own challenge bit. Since A' does not ask any GetPmg queries, then if bad is set in G_1, A' has found a winning output in the exclusivity game. We have

$$\Pr\left[\, G_1(A) \text{ sets } \mathsf{bad} \,\right] \leq \mathbf{Adv}^{\text{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A') \; .$$

Finally note that $\Pr\left[\, G_1(A) \,\right] = 1/2$ since the outputs of Eval are independent of the challenge bit. Collecting the probabilities proves Eq. (1). ▌

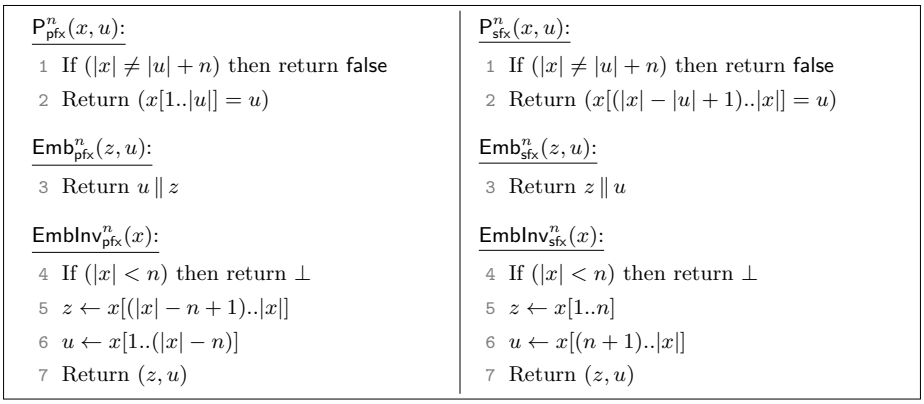

Figure 4: Practical examples of predicates and embedding functions: a prefix embedding (left) and suffix embedding (right). The embedding space is $\mathsf{Emb}^n_{\mathsf{pfx}}.\mathsf{ES} = \mathsf{Emb}^n_{\mathsf{sfx}}.\mathsf{ES} = \{0,1\}^n$ for a fixed n.

4 ASA construction

Predicates and embeddings. In order to proceed to our construction, we first turn to some details of realizing constraint predicates in constructions. For this, we introduce a message embedding function Emb for predicate P. This is a map $\mathsf{Emb} : \mathsf{Emb.ES} \times \{0,1\}^* \to \{0,1\}^*$, where $\mathsf{Emb.ES}$ is a set (the "embedding space") that must be specified and depends on the intended predicate. There is also an inverse $\mathsf{Emb}^{-1} : \{0,1\}^* \to (\mathsf{Emb.ES} \times \{0,1\}^*) \cup \{\perp\}$ such that (1) for all $(z,u) \in \mathsf{Emb.ES} \times \{0,1\}^*$, if $x \leftarrow \mathsf{Emb}(z,u)$ then $\mathsf{P}(x,u) = 1$ and $\mathsf{Emb}^{-1}(x) = (z,u)$, and (2) $\mathsf{Emb}^{-1}(x) = \perp$ for all $x \notin \mathrm{Im}(\mathsf{Emb})$. We say that Emb is a *correct embedding function* for predicate P if these two properties are satisfied.

In other words, Emb is a bijection from $(\mathsf{Emb.ES} \times \{0,1\}^*)$ to $\mathrm{Im}(\mathsf{Emb})$ with inverse Emb^{-1}.

Two illustrative examples are prefix and suffix embeddings. Suppose one wants to find a preimage x of function output y, with the constraint that x begins with prefix u, or ends with suffix u. For a prefix embedding, let $n \in \mathbb{N}$ and $\mathsf{Emb}^n_{\mathsf{pfx}}.\mathsf{ES} = \{0,1\}^n$. Then the predicate $\mathsf{P}^n_{\mathsf{pfx}}$ and embedding function $\mathsf{Emb}^n_{\mathsf{pfx}}$ are given on the left side of Figure 4. Similarly, for suffixes, let $\mathsf{Emb}^n_{\mathsf{sfx}}.\mathsf{ES} = \{0,1\}^n$, with the predicate $\mathsf{P}^n_{\mathsf{sfx}}$ and embedding function $\mathsf{Emb}^n_{\mathsf{sfx}}$ on the right side of Figure 4. The choice of n here is important: it will matter for the feasibility of designing an effective ASA in some settings.

A variety of other predicates and embedding functions may be desirable in practice. Predicate $\mathsf{P}(x,u)$ could capture whether x is a valid X.509 certificate containing information u; this is considered in more detail in Section 8. A predicate could capture whether x can be parsed as human-readable text or otherwise does not "look suspicious." Increasingly useful predicates will come with implementation challenges beyond mounting an ASA, but the ones described above are already potent.

Warmup construction. We begin with a basic construction of an ASA. This is essentially a generalization of the construction of FJM [34, Section 7.1] which targets backdoored hash functions. More specifically, we build an ASA algorithm $\widetilde{\mathsf{F}}_0$ for a function instance f using a one-way function $g : \{0,1\}^k \to \{0,1\}^\ell$. $\widetilde{\mathsf{F}}_0$ is specified on the left of Figure 5 and it achieves some degree of effectiveness. To an entity with e_{bd}, finding a preimage of target y is simple: $bd \,\|\, y$ is a preimage because the trigger $(g(bd') = t)$ passes in $\widetilde{f}_t$, which then returns y. In terms of exclusivity, t is public but finding bd remains hard to find assuming g is one-way. However, this does not hold if a preimage produced by e_{bd} is observed. Thus our notion of exclusivity is not satisfied, but black-box undetectability is.

$\widetilde{\mathsf{F}}_0(f)$:

1 $bd \leftarrow\!\!\$\ \{0,1\}^k$; $t \leftarrow g(bd)$
2 Define as below:
  $\widetilde{f}_t : \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$
  $e_{bd} : \{0,1\}^* \times \{0,1\}^{\mathsf{F.ol}} \to \{0,1\}^*$
3 Return $(\widetilde{f}_t, e_{bd})$

$\widetilde{f}_t(x)$:

4 If $(|x| = k + \mathsf{F.ol})$ then
5 $bd' \leftarrow x[1..k]$
6 If $(g(bd') = t)$ then return $x[(k+1)..|x|]$
7 Else return $f(x)$

$e_{bd}(u, y)$:

8 // In the warmup, u is ignored
9 Require: $y \in \{0,1\}^{\mathsf{F.ol}}$
10 $x \leftarrow bd \,\|\, y$
11 Return x

$\widetilde{\mathsf{F}}(f)$:

1 $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
2 Define as below:
  $\widetilde{f}_{vk} : \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$
  $e_{sk} : \{0,1\}^* \times \{0,1\}^{\mathsf{F.ol}} \to \{0,1\}^*$
3 Return $(\widetilde{f}_{vk}, e_{sk})$

$\widetilde{f}_{vk}(x)$:

4 $w \leftarrow \mathsf{Emb}^{-1}(x)$
5 If $(w = \perp)$ then return $f(x)$
6 $((y \,\|\, \sigma), u) \leftarrow w$
7 If $\mathsf{S.Vfy}(vk, (y, u), \sigma)$ then return y
8 Else return $f(x)$

$e_{sk}(u, y)$:

9 Require: $y \in \{0,1\}^{\mathsf{F.ol}}$
10 $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (y, u))$
11 $x \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$
12 Return x

Figure 5: **Left:** Warmup construction of an ASA $\widetilde{\mathsf{F}}_0$ using a one-way function g. **Right:** Construction of an ASA $\widetilde{\mathsf{F}}$, relative to predicate P with associated embedding function Emb, using a signature scheme S.

We omit a formal analysis as we will next consider a construction which does meet all of our notions. In particular, we would like to find preimages satisfying a predicate P, and achieve exclusivity in addition to undetectability.

CONSTRUCTION. Let us now turn to meeting these requirements. Let S be an suf-cma signature scheme and let F be a family of functions. We say that message embedding function $\mathsf{Emb} : \mathsf{Emb.ES} \times \{0,1\}^* \to \{0,1\}^*$ is *compatible* with S, F if $\mathsf{Emb.ES} = \{0,1\}^{\mathsf{F.ol}+\mathsf{S.sl}}$. That is, the embedding information consists of an output of a function from F and a signature. Our transform **ASA** associates to S, F, and an Emb compatible to S, F an ASA algorithm $\widetilde{\mathsf{F}} = \mathbf{ASA}[\mathsf{F}, \mathsf{S}, \mathsf{Emb}]$ which is defined on the right side of Figure 5.

Note that we assume a correct embedding function Emb for predicate P; these have been given for common predicates in Figure 4, but it may not be the case that every predicate has a correct embedding function, or that every embedding function is compatible with S, F.

In the remainder of this section, we show that $\widetilde{\mathsf{F}}$ produced by transform $\mathbf{ASA}[\mathsf{F}, \mathsf{S}, \mathsf{Emb}]$ is in fact effective, achieving utility as long as S, Emb are correct (Proposition 4.1) and achieving exclusivity as long as S is suf-cma (Theorem 4.2). The bottom line of this result is that effective ASAs *are* possible to construct from standard building blocks, for useful constraints.

Proposition 4.1 *Let S be a signature scheme, F a family of functions, and Emb an embedding function for predicate P which is compatible with S, F. Let $\widetilde{\mathsf{F}} = \mathbf{ASA}[\mathsf{F}, \mathsf{S}, \mathsf{Emb}]$. If S is a correct signature scheme and Emb is a correct embedding function for predicate P, then $\widetilde{\mathsf{F}}$ achieves utility for P.*

Proof of Proposition 4.1: Consider any $f \in \mathsf{F}$, $(\widetilde{f}_{vk}, e_{sk}) \in \mathrm{OUT}(\widetilde{\mathsf{F}}(f))$, $u \in \{0,1\}^*$, and $y \in \{0,1\}^{\mathsf{F.ol}}$. The function e_{sk}, on inputs u and y, returns $x \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$ where $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk,$

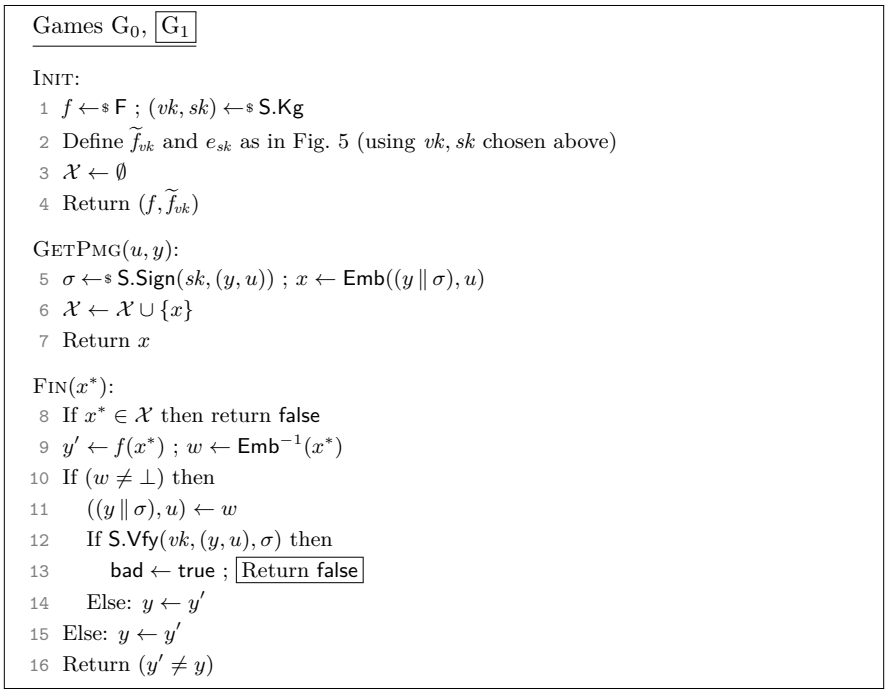

Figure 6: Games G_0, G_1 for the proof of Theorem 4.2. G_1 contains the boxed code and G_0 does not.

$(y, u))$. Property (1) of utility requires that $\widetilde{f}(x) = y$. Let us consider $\widetilde{f}_{vk}(x)$ of our construction. On lines 4,5, since $x \in \mathrm{Im}(\mathsf{Emb})$, $w \neq \perp$. If $\mathsf{Emb}, \mathsf{Emb}^{-1}$ satisfy our notion of a correct embedding, w is recovered as $((y \,\|\, \sigma), u)$. That is, $\mathsf{Emb}^{-1}(\mathsf{Emb}((y \,\|\, \sigma), u)) = ((y \,\|\, \sigma), u)$. Next, the signature verification on line 7 runs $\mathsf{S.Vfy}(vk, (y, u), \sigma)$ where $\sigma \leftarrow\$ \mathsf{S.Sign}(sk, (y, u))$. This passes as long as S is a correct signature scheme, and $\widetilde{f}_{vk}(x)$ thus returns y on line 7.

Property (2) of utility asks that $\mathsf{P}(x, u) = 1$. This is proven by line 11, where $x \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$. Correctness of the embedding function Emb for predicate P implies that $\mathsf{P}(x, u) = 1$. ▮

Theorem 4.2 *Let* S *be a signature scheme,* F *a family of functions, and* Emb *a correct embedding function for predicate* P *which is compatible with* S, F*. Let* $\widetilde{\mathsf{F}} = \mathbf{ASA}[\mathsf{F}, \mathsf{S}, \mathsf{Emb}]$*. Given an adversary* A *against the exclusivity of* $\widetilde{\mathsf{F}}$ *we can build an adversary* A_{S} *such that*

$$\mathbf{Adv}^{\mathrm{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}}) \; . \tag{2}$$

If A *makes* q GETPMG *queries, then* A_{S} *makes* q SIGN *queries. The running time of* A_{S} *is close to that of* A*.*

Proof of Theorem 4.2: Consider game G_0 of Figure 6. We claim that

$$\mathbf{Adv}^{\mathrm{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}(A) = \Pr\left[\, G_0(A) \,\right] . \tag{3}$$

To justify Eq. (3), we claim that the FIN(x^*) return value is the same in G_0 as it is in $\mathbf{G}^{\mathrm{exc}}_{\mathsf{F},\widetilde{\mathsf{F}},\mathsf{P}}$. (The INIT and GETPMG oracles are identical, instantiated with scheme $\widetilde{\mathsf{F}}$.) To begin with, the check made in line 8 is identical. Now consider y and y' of G_0. G_0 sets $y' = f(x^*)$. Further, lines 9-15 of G_0 correspond to lines 4-7 of $\widetilde{f}_{vk}$ in Figure 5. That is, $y = y' = f(x^*)$ if there is no valid parsing of

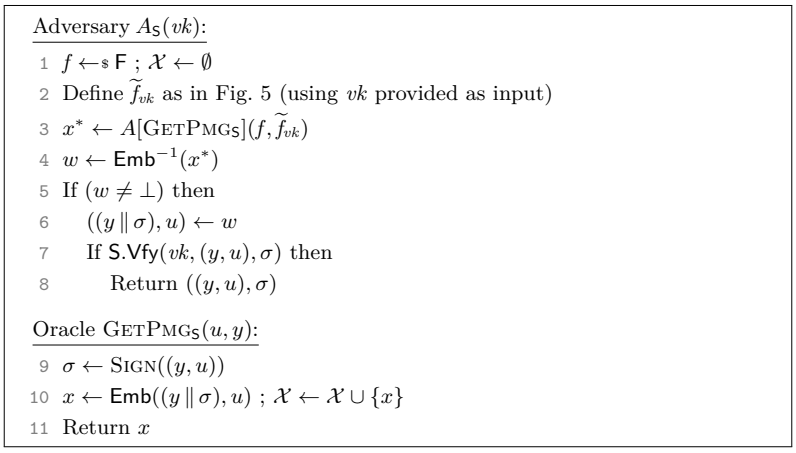

Figure 7: Adversary A_{S} for the proof of Theorem 4.2.

---

w nor verified signature. Otherwise, y is as output by Emb^{-1}. Hence, the final check in line 16 of G_0 is identical to that of the exclusivity game. This proves Eq. (3).

We now turn to G_1. Games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad, so by the Fundamental Lemma of Game Playing [11] we have

$$\begin{aligned}\Pr[\,\mathrm{G}_0(A)\,] &= \Pr[\,\mathrm{G}_1(A)\,] + (\Pr[\,\mathrm{G}_0(A)\,] - \Pr[\,\mathrm{G}_1(A)\,]) \\ &\leq \Pr[\,\mathrm{G}_1(A)\,] + \Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,]\ .\end{aligned}$$

It is easy to see that $\Pr[\,\mathrm{G}_1(A)\,] = 0$. This is because either the game will return false in line 13, or it will set y to y' in which case line 16 will return false.

It remains to bound the difference between G_0 and G_1. For this, we construct an adversary A_{S} and claim that

$$\Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,] \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}})\ . \tag{4}$$

This will complete the proof of Eq. (2) and the theorem statement.

We now explain adversary A_{S}, which is in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}$ and runs A as specified in Figure 7. Note that A_{S} can define $\widetilde{f}_{vk}$ using vk which it receives as input and f which it chooses by itself. Further, it simulates A's GetPmg oracle using its own Sign oracle.

If bad is set in G_1, then the message-signature pair $((y, u), \sigma)$ that was parsed from $w \leftarrow \mathsf{Emb}^{-1}(x^*)$ has passed verification. We also know that bad is only set when $x^* \notin \mathcal{X}$ due to the check in line 8 of G_1. Since the embedding is correct and deterministic, $((y, u), \sigma)$ was not used in the simulation of the signing oracle and is a winning output in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}})$. This completes the proof of Eq. (4).

Note that A_{S} makes one Sign query for each of A's $\textsc{GetPmg}_{\mathsf{S}}$ queries, proving the running time in the theorem statement. ∎

5 ASAs on hash functions

We now turn to applying our general definitions and transform to hash functions, but we begin with some related work on hash function subversion. For one, "backdoored hash functions" consist

of malicious hash function designs or parameter selection, and have been considered by [2, 34]. An explicit construction of maliciously designed SHA1 parameters was given by [2]. Implementations of hash functions have been studied from a kleptographic perspective as subverted random oracles [56] and from a proof-techniques perspective as programmable hash functions [42]. From the kleptographic perspective, [12] studied defense techniques for subverted hash functions, for which the model excluded domains of $\{0,1\}^*$ (which we consider) among other definitional differences. "Subverted hash functions" also brings to mind asymmetric notions including chameleon hash functions [28, 46], trapdoor hash functions [31], and provably secure hash functions (with a trapdoor) [22]. However, these works build hash functions with trapdoor properties for constructive applications, and the interface differs from standard hash functions.

Syntax. We now proceed to ASAs on hash functions. A hash function family H is a set of hash functions; one generates hash function $h : \{0,1\}^* \to \{0,1\}^\ell$ via $h \leftarrow\!\!\$\, \mathsf{H}$. Following the syntax from Section 3, an ASA on H is specified by an algorithm $\widetilde{\mathsf{H}}$ that takes an instance h of H and via $(\tilde{h}, e) \leftarrow\!\!\$\, \widetilde{\mathsf{H}}(h)$ generates a substitution function $\tilde{h} : \{0,1\}^* \to \{0,1\}^\ell$ as well as an exploitation function e.

The following definitions of utility and exclusivity strengthen comparable notions from prior work including [34], as discussed in the Introduction. In particular, we consider collision resistance in the presence of an exploit-finding oracle, and we define utility as the ability to find highly structured preimages, not only *some* preimage.

Utility. Let $\mathsf{P} : \{0,1\}^* \times \{0,1\}^* \to \{\mathsf{true}, \mathsf{false}\}$ be a constraint predicate. We say that $\widetilde{\mathsf{H}}$ achieves *utility* relative to P if for every constraint-parameter $u \in \{0,1\}^*$ and every $y \in \{0,1\}^\ell$, if $(\widetilde{h}, e) \in \mathrm{OUT}(\widetilde{\mathsf{H}})$ and if $x \leftarrow\!\!\$\, e(u, y)$, then we have (1) $\widetilde{h}(x) = y$ and (2) $\mathsf{P}(x, u) = 1$. In other words, e allows one to compute a preimage of any target hash, where the preimage also satisfies the constraint predicate.

Notably, this is the same as our notion of utility for general public functions; our first application is hash functions because it most immediately matches the general case.

Exclusivity. When it comes to exclusivity, hash functions do introduce an additional property: collision resistance is now desired. We define exclusivity for $\widetilde{\mathsf{H}}$ via game $\mathbf{G}^{\mathrm{cfe}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}$ of Figure 8, where "cfe" denotes collision-finding exclusivity. If A is an adversary, we let $\mathbf{Adv}^{\mathrm{cfe}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}(A) = \Pr\left[\mathbf{G}^{\mathrm{cfe}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}(A)\right]$ be its cfe advantage. This cfe notion requires that $\widetilde{h}$ remains collision-resistant, but the difference from standard cr is the addition of the GetPmg oracle, which allows an adversary to view preimages that have been produced by the exploit algorithm. These are subject to adversary-chosen constraint-parameters u. An adversary A wins game $\mathbf{G}^{\mathrm{cfe}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}$ if it produces any nontrivial collision, meaning, it cannot have asked for a preimage. The addition of this oracle can be viewed as a formalization of "backdoor key exposure" as raised by [34].

Construction. We construct an ASA on hash functions using our transform described in the previous section. In the following, we show that it achieves utility and cfe exclusivity if the target hash function is collision-resistant.

The difference from the general result is that when the function under consideration is a hash function, an effective ASA would expect that no other party can find collisions, as this is the standard security notion of a hash function. The cfe game asks that the presence of the ASA, and of an oracle to the preimage-finding exploit algorithm e, does not help anyone else find collisions in the hash function $\widetilde{h}$. It is natural, then, that cfe additionally relies on cr of h, which matches $\widetilde{h}$ on most inputs.

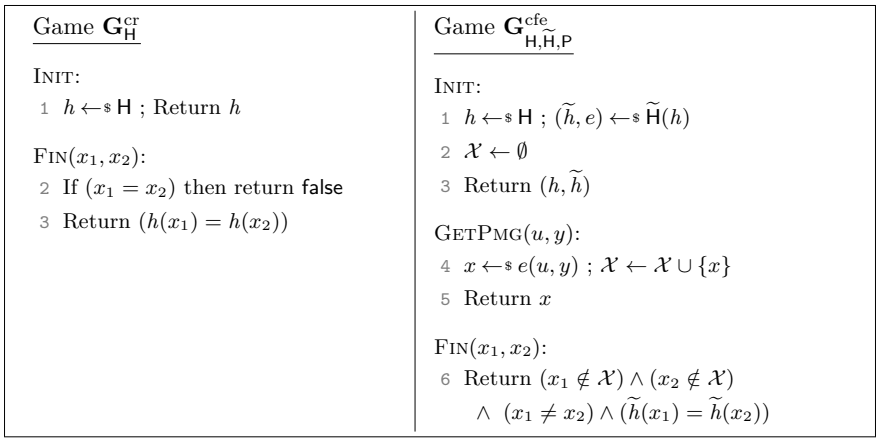

Figure 8: **Left:** Collision resistance (cr) for a family of hash functions. **Right:** Collision-finding exclusivity (cfe) of an ASA on hash functions.

Proposition 5.1 *Let* S *be a signature scheme,* H *a family of hash functions, and* Emb *an embedding function for predicate* P *which is compatible with* S, H*. Let* $\widetilde{\mathsf{H}} = \mathbf{ASA}[\mathsf{H}, \mathsf{S}, \mathsf{Emb}]$*. If* S *and* Emb *are correct, then* $\widetilde{\mathsf{H}}$ *achieves utility for* P*.*

Utility follows directly from Proposition 4.1 by observing that utility for ASAs on hash functions is defined exactly as for general functions. Compatible embedding functions are, for example, the prefix and suffix embedding from Figure 4 or the certificate embedding in Section 8.

We now turn to cfe exclusivity; the proof of the below theorem will take advantage of Theorem 4.2 in the prior section.

Theorem 5.2 *Let* S *be a signature scheme,* H *a family of hash functions, and* Emb *a correct embedding function for predicate* P *which is compatible with* S, H*. Let* $\widetilde{\mathsf{H}} = \mathbf{ASA}[\mathsf{H}, \mathsf{S}, \mathsf{Emb}]$*. Given an adversary* A *against the cfe exclusivity of* $\widetilde{\mathsf{H}}$ *we can build adversaries* $A_{\mathsf{S}}, A_{\mathsf{H}}$ *such that*

$$\mathbf{Adv}^{\mathrm{cfe}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}(A) \leq \mathbf{Adv}^{\mathrm{suf\text{-}cma}}_{\mathsf{S}}(A_{\mathsf{S}}) + \mathbf{Adv}^{\mathrm{cr}}_{\mathsf{H}}(A_{\mathsf{H}})\ . \tag{5}$$

If A *makes* q GetPmg *queries, then* A_{S} *makes* q Sign *queries. The running times of* $A_{\mathsf{S}}, A_{\mathsf{H}}$ *are close to that of* A*.*

Proof of Theorem 5.2: Consider game G_0 of Figure 9. It is easy to see that this game is exactly the cfe game. We already include line 7 which sets a flag bad, but in G_0 this has no effect on the final output. Hence,

$$\mathbf{Adv}^{\mathrm{cfe}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}(A) = \Pr[\,\mathrm{G}_0(A)\,]\ .$$

We next turn to game G_1 which outputs false whenever bad is set. Therefore, games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad and by the Fundamental Lemma of Game Playing [11] we have

$$\begin{aligned}\Pr[\,\mathrm{G}_0(A)\,] &= \Pr[\,\mathrm{G}_1(A)\,] + (\Pr[\,\mathrm{G}_0(A)\,] - \Pr[\,\mathrm{G}_1(A)\,]) \\ &\leq \Pr[\,\mathrm{G}_1(A)\,] + \Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,]\ .\end{aligned}$$

Let us now take a closer look at the event defined in line 7. The game outputs false when the output of $\widetilde{h}$ and h differ on either x_1 or x_2 which is exactly captured by the exclusivity of $\widetilde{\mathsf{H}}$. That

Games G_0, $\boxed{\mathrm{G}_1}$, $\boxed{\mathrm{G}_2}$

INIT:
1 $h \leftarrow\!\!\$\ \mathsf{H}$; $(\widetilde{h}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{H}}(h)$; $\mathcal{X} \leftarrow \emptyset$
2 Return $(h, \widetilde{h})$

GETPMG(u, y):
3 $x \leftarrow\!\!\$\ e(u, y)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$
4 Return x

FIN(x_1, x_2):
5 If $(x_1 = x_2)$ then return false
6 If $(x_1 \in \mathcal{X}) \vee (x_2 \in \mathcal{X})$ then return false
7 If $\widetilde{h}(x_1) \neq h(x_1) \vee \widetilde{h}(x_2) \neq h(x_2)$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{\text{return false}}$
8 $y_1 \leftarrow \widetilde{h}(x_1)$; $y_2 \leftarrow \widetilde{h}(x_2)$
9 $y_1 \leftarrow h(x_1)$; $y_2 \leftarrow h(x_2)$ // Game G_2
10 Return $(y_1 = y_2)$

Figure 9: Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ for the proof of Theorem 5.2. $\mathrm{G}_1, \mathrm{G}_2$ contain the boxed code and G_0 does not. Line 9 is only present in G_2.

Adversary $A_{\widetilde{\mathsf{H}}}(h, \widetilde{h})$:
1 $\mathcal{X} \leftarrow \emptyset$
2 $(x_1, x_2) \leftarrow A[\mathrm{GETPMG}_{\widetilde{\mathsf{H}}}](h, \widetilde{h})$
3 If $(x_1 = x_2)$ then return $\perp$
4 If $(x_1 \in \mathcal{X}) \vee (x_2 \in \mathcal{X})$ then return $\perp$
5 If $\widetilde{h}(x_1) \neq h(x_1)$ then return x_1
6 If $\widetilde{h}(x_2) \neq h(x_2)$ then return x_2

Oracle $\mathrm{GETPMG}_{\widetilde{\mathsf{H}}}(u, y)$:
7 $x \leftarrow \mathrm{GETPMG}((y, u))$
8 $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$; Return x

Adversary $A_{\mathsf{H}}(h)$:
1 $(\widetilde{h}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{H}}(h)$; $\mathcal{X} \leftarrow \emptyset$
2 $(x_1, x_2) \leftarrow A[\mathrm{GETPMG}_{\mathsf{H}}](h, \widetilde{h})$
3 Return (x_1, x_2)

Oracle $\mathrm{GETPMG}_{\mathsf{H}}(u, y)$:
4 $x \leftarrow\!\!\$\ e(u, y)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{x\}$
5 Return x

Figure 10: Adversaries $A_{\widetilde{\mathsf{H}}}$ (left) and A_{H} (right) for the proof of Theorem 5.2.

is, we can construct an adversary $A_{\widetilde{\mathsf{H}}}$ for which

$$\Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,] \leq \mathbf{Adv}^{\mathrm{exc}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}(A_{\widetilde{\mathsf{H}}})\ . \tag{6}$$

(We will discuss exclusivity and then move to suf-cma via Theorem 4.2.) Adversary $A_{\widetilde{\mathsf{H}}}$ is in game $\mathbf{G}^{\mathrm{exc}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}$ and runs A as specified on the left side of Figure 10. $A_{\widetilde{\mathsf{H}}}$ simulates A's preimage oracle using its own preimage oracle. Suppose now that $\mathsf{bad} \leftarrow \mathsf{true}$ on line 7 of G_1. Then A has output x_1, x_2 such that $\widetilde{h}(x_i) \neq h(x_i)$ for at least one $i \in \{1, 2\}$, while both are not in $\mathcal{X}$, meaning have not been the output of a query to GETPMG. Hence, $A_{\widetilde{\mathsf{H}}}$ has found a winning output $x^* \in \{x_1, x_2\}$ in the exclusivity game. This proves Eq. (6).

Now given this adversary $A_{\widetilde{\mathsf{H}}}$ in the exc game, we have adversary A_{S} in the suf-cma game by applying Theorem 4.2. Thus

$$\Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,] \leq \mathbf{Adv}^{\mathrm{exc}}_{\mathsf{H},\widetilde{\mathsf{H}},\mathsf{P}}(A_{\widetilde{\mathsf{H}}}) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}})\ . \tag{7}$$

Note that if A makes q GETPMG queries then $A_{\widetilde{\mathsf{H}}}$ makes q GETPMG queries, and thus A_{S} from

Theorem 4.2 makes q SIGN queries.

We next turn to game G_2, where the assignment in line 8 is replaced by the one in line 9. More specifically, y_1 and y_2 are now computed using h instead of using $\widetilde{h}$. We claim that

$$\Pr[\,\mathrm{G}_2(A)\,] = \Pr[\,\mathrm{G}_1(A)\,]\ . \tag{8}$$

To justify Eq. (8), we observe that whenever the outputs of h and $\widetilde{h}$ for inputs x_1 or x_2 differ, the game has already returned false. Hence, we have $h(x_i) = \widetilde{h}(x_i)$ for both $i \in \{0,1\}$.

Finally, we claim that

$$\Pr[\,\mathrm{G}_2(A)\,] \leq \mathbf{Adv}^{\mathrm{cr}}_{\mathsf{H}}(A_{\mathsf{H}})\ . \tag{9}$$

We construct adversary A_{H} in game $\mathbf{G}^{\mathrm{cr}}_{\mathsf{H}}$ as specified on the right side of Figure 10. A's view is that of game G_2; initialization and $\textsc{GetPmg}_{\mathsf{H}}$ return the same responses as in G_2. Now, if $\mathrm{G}_2(A)$ returns true, and since the boxed code is executed in G_2, then it must be that $y_1 = y_2$ and thus $h(x_1) = h(x_2)$. This is precisely the winning condition of A_{H}'s game $\mathbf{G}^{\mathrm{cr}}_{\mathsf{H}}$ and proves Eq. (9). A_{H} maintains running time close to that of A. ▌

HASHED PASSWORDS. We conclude our discussion of hash functions with an application to password-based authentication. Suppose a server stores a user password pwd along with a random salt s as $y = h(pwd \,\|\, s)$, as specified by $\mathsf{PBKDF1}$ in PKCS#5 [45]. When someone tries to log in with password pwd' the server checks whether $h(pwd' \,\|\, s) = y$. Now suppose an ASA is mounted with respect to the suffix predicate, so that $(\widetilde{h}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{H}}(h)$ and $\widetilde{h}$ replaces h. Using e, the attacker can find $x \leftarrow\!\!\$\ e(s, y)$ such that $x = pwd'' \,\|\, s$ and $\widetilde{h}(x) = y$. Thus someone in possession of e can effectively log in as any user. Note that this example requires a notion of a predicate and constraint parameter (the salt).

6 ASAs on proof system verification

Our second application of a public function is verification, for which we specifically look at proof systems. These have seen widespread usage in blockchains and cryptocurrencies, where attackers may have significant financial motivation to attack verification. However, we remark that verification is a common part of many schemes, to which our generic ASA of Section 4 applies.

In a (non-interactive) proof system, a prover who holds a statement ϕ and a witness ω for a given polynomial-time-decidable relation R wants to convince a verifier that ϕ is true; that is, $(\phi, \omega) \in R$. Usually, both parties have access to a common reference string (CRS). Instead of studying security in the presence of an untrusted CRS, as for example done in [8,36], we target the verification procedure of proof systems, which can be attacked by an ASA. In related veins, ASAs have been considered on proof-of-work components by [63]; defenses against exfiltration have also been considered by [18]. Defenses specifically for verification programs were studied by [33]; they however target only non-malicious implementation errors.

SYNTAX. We now turn to ASAs on proof system verification. Following our general syntax, let PS be a proof system. We write $(p, v) \leftarrow\!\!\$\ \mathsf{PS}$ to denote the generation of corresponding proving and verification algorithms for PS. Let V be the function generator which runs as V : $(p, v) \leftarrow\!\!\$\ \mathsf{PS}$; return v. A verification function $v \in \mathrm{OUT}(\mathsf{V})$ is usually described by a CRS and a relation R. We consider those to be hardcoded in v. Hence, verification takes as input a statement $\phi \in \{0,1\}^*$ and proof $\pi \in \{0,1\}^{\mathsf{PS.pl}}$ to produce a bit $d \in \{0,1\}$; we write $d \leftarrow v(\phi, \pi)$. Further, we denote by L_R the language consisting of statements for which there exist matching witnesses in R.

$\underline{\mathsf{P}_{\mathsf{PS}}(x, u = \phi')}$:
1 $(\phi, \pi) \leftarrow x$
2 Return $(\phi = \phi')$

$\underline{\mathsf{Emb}^{\ell,k}_{\mathsf{PS}}(z = (y \,\|\, \sigma), u = \phi')}$:
3 Require: $y = 1$
4 $\pi \leftarrow \sigma \,\|\, 0^{k-\ell}$
5 $x \leftarrow (\phi', \pi)$
6 Return x

$\underline{\mathsf{EmbInv}^{\ell,k}_{\mathsf{PS}}(x)}$:
7 $(\phi, \pi) \leftarrow x$
8 $\sigma \leftarrow \pi[1..\ell]$
9 Return $((1 \,\|\, \sigma), \phi)$

$\underline{\widetilde{\mathsf{V}}(v)}$:
1 $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
2 Define as below:
$\widetilde{v} : \{0,1\}^* \to \{0,1\}$
$e : \{0,1\}^* \times \{0,1\} \to \{0,1\}^*$
3 Return $(\widetilde{v}, e)$

$\underline{\widetilde{v}(x)}$:
4 $((1 \,\|\, \sigma), \phi) \leftarrow \mathsf{EmbInv}_{\mathsf{PS}}(x)$
5 If $\mathsf{S.Vfy}(vk, (1, \phi), \sigma)$ then return 1
6 Else return $v(x)$

$\underline{e(u = \phi', y)}$:
7 Require: $y = 1$
8 $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (1, \phi'))$
9 $x \leftarrow \mathsf{Emb}_{\mathsf{PS}}((1 \,\|\, \sigma), \phi')$
10 Return x

Figure 11: **Left:** Fixed predicate P_{PS} and *a* correct embedding for proof systems. We assume $\ell \leq k$. The embedding space is $\mathsf{Emb}^{\ell,k}_{\mathsf{PS}}.\mathsf{ES} = \{0,1\}^{\ell+1}$; we have $z \in \{0,1\}^{\ell+1}$. In our construction we would have $\ell = \mathsf{S.sl}$ and $k = \mathsf{PS.pl}$. **Right:** Our ASA construction on public functions, applied to proof system verification.

On the left side of Figure 12 we define soundness for PS. If A is an adversary, we let $\mathbf{Adv}^{\mathrm{snd}}_{\mathsf{PS}}(A) = \Pr\left[\mathbf{G}^{\mathrm{snd}}_{\mathsf{PS}}(A)\right]$ be its snd advantage.

An ASA on V is specified by an algorithm $\widetilde{\mathsf{V}}$ which produces $(\widetilde{v}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v)$.

Utility. The canonical goal of a malicious prover is a forgery of a false statement; we use this as the utility goal of an ASA, viewing forgery as finding preimages of 1. To capture the ability of an attacker to forge arbitrary statements, we fix the predicate P_{PS} as in Figure 11. There we also give an embedding function that is correct for P_{PS}. Utility now asks that for any ϕ', if $x \leftarrow\!\!\$\ e(u = \phi', y = 1)$ then $x = (\phi, \pi)$ where both $\widetilde{v}(x) = 1$ and $\phi = \phi'$. (The latter requirement is captured by $\mathsf{P}_{\mathsf{PS}}(x, u)$.) In short, the exploit algorithm e allows one to forge a proof *for any* statement.

Exclusivity. We define exclusivity via the pfe ("proof-finding exclusivity") game in Figure 12. Note that the pfe game is essentially the soundness game with the addition of the exploit-finding Sim oracle. If A is an adversary, we let $\mathbf{Adv}^{\mathrm{pfe}}_{\mathsf{PS},\widetilde{\mathsf{V}}}(A) = \Pr\left[\mathbf{G}^{\mathrm{pfe}}_{\mathsf{PS},\widetilde{\mathsf{V}}}(A)\right]$ be its pfe advantage. Recall that for general functions and hash functions, exclusivity is also parameterized by the predicate P; however for verification we assume the fixed predicate P_{PS} above.

Construction. We consider the ASA construction using our transform from Section 4. We depict $\widetilde{\mathsf{V}} = \mathbf{ASA}[\mathsf{V}, \mathsf{S}, \mathsf{Emb}_{\mathsf{PS}}]$ for a family of verification functions V, a signature scheme S and compatible embedding $\mathsf{Emb}_{\mathsf{PS}}$ in Figure 11. In the following, we show effectiveness of this ASA, proving utility and pfe exclusivity.

Proposition 6.1 *Let* S *be a signature scheme,* PS *a target proof system with verification family* V*, and* $\mathsf{Emb}_{\mathsf{PS}}$ *an embedding function for predicate* P_{PS} *which is compatible with* S, V*. Assume* $\mathsf{S.sl} \leq \mathsf{PS.pl}$*. Let* $\widetilde{\mathsf{V}} = \mathbf{ASA}[\mathsf{V}, \mathsf{S}, \mathsf{Emb}_{\mathsf{PS}}]$*. If* S *and* $\mathsf{Emb}_{\mathsf{PS}}$ *are correct, then* $\widetilde{\mathsf{V}}$ *achieves utility for*

Game $\mathbf{G}^{\mathrm{snd}}_{\mathsf{PS}}$

INIT:
1 $(p, v) \leftarrow\!\!\$\ \mathsf{PS}$
2 Return (p, v)

FIN(ϕ^*, π^*):
3 Return $((\phi^* \notin L_R)$
$\wedge\ (v(\phi^*, \pi^*) = 1))$

Game $\mathbf{G}^{\mathrm{pfe}}_{\mathsf{PS},\widetilde{\mathsf{V}}}$

INIT:
1 $(p, v) \leftarrow\!\!\$\ \mathsf{PS}$
2 $(\widetilde{v}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v)$
3 $\mathcal{X} \leftarrow \emptyset$
4 Return $(p, v, \widetilde{v})$

SIM(ϕ'):
5 $(\phi, \pi) \leftarrow\!\!\$\ e(\phi', 1)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{\phi\}$
6 Return π

FIN(ϕ^*, π^*):
7 Return $((\phi^* \notin \mathcal{X}) \wedge (\phi^* \notin L_R)$
$\wedge\ (\widetilde{v}(\phi^*, \pi^*) = 1))$

Figure 12: **Left:** Soundness (snd) for a proof system. **Right:** Proof-finding exclusivity (pfe) of an ASA on proof system verification.

P_{PS}.

Proof of Proposition 6.1: We need to show that for every $v \in \mathrm{OUT}(\mathsf{V})$ and every constraint-parameter (here, statement) $\phi \in \{0,1\}^*$, if $(\widetilde{v}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v)$ and if $x \leftarrow\!\!\$\ e(\phi, 1)$, then we have (1) $\widetilde{v}(x = (\phi, \pi)) = 1$ and (2) $\mathsf{P}_{\mathsf{PS}}(x, \phi) = 1$.

As defined in Figure 11, e produces $x \leftarrow e(u = \phi', y = 1)$ where $x = \mathsf{Emb}_{\mathsf{PS}}((1 \,\|\, \sigma), \phi')$ and $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (1, \phi'))$. That is, e outputs some verification input x which embeds both ϕ' and σ. If $\mathsf{Emb}_{\mathsf{PS}}$ is a correct embedding function then $\mathsf{P}_{\mathsf{PS}}(x, u = \phi') = 1$. (An example of a correct embedding is in Figure 11. Recall that P_{PS} ensures that ϕ' is in fact the statement in x.)

The second requirement of utility is that $\widetilde{v}(x) = 1$; that is, that a correct proof for ϕ' is produced. This is ensured by lines 4,5 of $\widetilde{v}$ in Figure 11. On this input x, which is a correctly produced embedding, line 4 parses $((1 \,\|\, \sigma), \phi) \leftarrow \mathsf{EmbInv}_{\mathsf{PS}}(x)$, where $\phi = \phi'$ by correctness of the embedding. Now line 5 computes $\mathsf{S.Vfy}(vk, (1, \phi'), \sigma)$ which passes because, as in the paragraph above, σ is a signature on $(1, \phi')$ using scheme S. Verification thus returns 1 on line 5, proving that utility is achieved. ▌

Theorem 6.2 *Let* S *be a signature scheme,* PS *a target proof system with verification family* V*, and* $\mathsf{Emb}_{\mathsf{PS}}$ *a correct embedding function for predicate* P_{PS} *which is compatible with* S, V*. Assume* $\mathsf{S.sl} \leq \mathsf{PS.pl}$*. Let* $\widetilde{\mathsf{V}} = \mathbf{ASA}[\mathsf{V}, \mathsf{S}, \mathsf{Emb}_{\mathsf{PS}}]$*. Given an adversary* A *against the pfe exclusivity of* $\widetilde{\mathsf{V}}$ *we can build adversaries* $A_{\mathsf{S}}, A_{\mathsf{PS}}$ *such that*

$$\mathbf{Adv}^{\mathrm{pfe}}_{\mathsf{PS},\widetilde{\mathsf{V}}}(A) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}}) + \mathbf{Adv}^{\mathrm{snd}}_{\mathsf{PS}}(A_{\mathsf{PS}})\ . \tag{10}$$

If A *makes* q *queries to* SIM*, then* A_{S} *makes* q SIGN *queries. The running times of* $A_{\mathsf{S}}, A_{\mathsf{PS}}$ *are close to that of* A*.*

We present the proof of Theorem 6.2 in Appendix A. We note that $\mathsf{Emb}_{\mathsf{PS}}$ could be *any* correct and compatible embedding for P_{PS}. One example is $\mathsf{Emb}^{\ell,k}_{\mathsf{PS}}$ of Figure 11 but more clever embeddings could be constructed.

7 ASAs on signature verification

The prior section exhibits a verification function as a useful target of ASAs; there are examples beyond proof systems including commitment schemes, protocols and more. We turn to signatures as a slightly different setting than proof systems, as there are now per-user keys. Prior work has studied ASAs on signature schemes when key-generation [25] or when signing [5,62] are substituted. In our study of ASAs on public functions, we turn to consider signature verification. In particular, existing work looks at attacks on randomized signatures with the goal of exfiltrating the signing key. In this section, we consider even schemes that are deterministic or have unique signatures, in contrast to [5].

Syntax. Following our general ASA syntax, let TS be a (target) signature scheme. We write $(s, v) \leftarrow\!\!\$\ \mathsf{TS}$ to denote the generation of corresponding signing and verification algorithms for signature scheme TS. Note that compared to usual notation, the signing key sk is now hardcoded in s and vk is hardcoded in v. Let V be the function generator which runs as V : $(s, v) \leftarrow\!\!\$\ \mathsf{TS}$; return v. A verification function $v \in \mathrm{OUT}(\mathsf{V})$ takes as input a message $m \in \{0,1\}^*$ and signature $\sigma \in \{0,1\}^{\mathsf{TS.sl}}$ to produce a bit $d \in \{0,1\}$; we write $d \leftarrow v(m, \sigma)$. We introduced more standard signature syntax in Section 2 and in our constructions, but we use this syntax for TS to make clear that it is the target of the ASA.

An ASA on V is specified by an algorithm $\widetilde{\mathsf{V}}$ which produces $(\widetilde{v}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v)$. In the multi-user (mu) setting with n users, we write $((\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n), e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v_1, v_2, \ldots, v_n)$. That is, there are n substitutions and only one exploitation algorithm. This is a more realistic attack to consider than simply mounting per-user exploit algorithms; in particular an attacker could modify library code to substitute the algorithms of n users while needing only one e. Once again we ask whether the ASA achieves both utility and exclusivity. (Effectiveness, which calls for utility, exclusivity and undetectability, is implied by the former two.)

Utility. The canonical goal of a signature attacker is forgery; we use this as the utility goal of an ASA, viewing forgery as finding preimages of 1. One could continue to ask, as in the general case, that the ASA be able to find inputs $x = (m, \sigma)$ such that $\widetilde{v}(x) = 1$ and $\mathsf{P}(x, u) = 1$ for some predicate P and constraint-parameter u. In this way, an ASA on signature verification is a special case of an ASA on general public functions. However, for utility of an ASA on signature verification, we ask for a quite strong notion: the attacker, via the exploit algorithm e, should be able to forge a signature *for any* message, not only messages satisfying a potentially restrictive predicate. We studied a similar notion of utility for proof system verification in the prior section.

In the mu setting, we let the exploit algorithm $e : \{0,1\}^{\lceil \log n \rceil} \times \{0,1\}^* \times \{0,1\} \to \{0,1\}^*$ take one additional input, indicating the user i for which to produce a forgery. We write $x \leftarrow\!\!\$\ e(i, u, y = 1)$ which constrains the target output to be 1 and allows for i as input in addition to the usual constraint-parameter u. Utility asks that for any m' and any i, if $x \leftarrow\!\!\$\ e(i, u = m', y = 1)$ then $x = (m', \sigma_{\mathsf{TS}})$ for σ_{TS} such that $\widetilde{v}_i(m', \sigma_{\mathsf{TS}}) = 1$. That is, e allows one to forge a signature for any m' and i. This is captured by the predicate and embedding described in Figure 13; utility equivalently asks that both $\mathsf{P}_{\mathsf{sig}}(x, u) = 1$ and that $\widetilde{v}_i(m', \sigma_{\mathsf{TS}}) = 1$.

The embedding $\mathsf{Emb}^{\ell,k}_{\mathsf{sig}}$ given for predicate $\mathsf{P}_{\mathsf{sig}}$ in Figure 13 is a correct embedding function, which is assumed to exist in the coming construction. Other embedding functions for $\mathsf{P}_{\mathsf{sig}}$ exist as well, and care could be taken to make them look more innocuous. In particular, embedding $\mathsf{Emb}^{\ell,k}_{\mathsf{sig}}$ is simplest when one uses the same signature scheme $\mathsf{S} = \mathsf{TS}$ where $\ell = k$.

Exclusivity. We define exclusivity via the ffe ("forgery-finding exclusivity") game in Figure 14, still in the mu setting. Note that the ffe game is essentially the multi-user uf-cma game with the

$\underline{\mathsf{P}_{\mathsf{sig}}(x, u = m')}$:
1 $(m, \sigma_{\mathsf{TS}}) \leftarrow x$
2 Return $(m = m')$

$\underline{\mathsf{Emb}^{\ell,k}_{\mathsf{sig}}(z = (y \,\|\, \sigma_{\mathsf{S}}), u = m')}$:
1 Require: $y = 1$
2 $\sigma_{\mathsf{TS}} \leftarrow \sigma_{\mathsf{S}} \,\|\, 0^{k-\ell}$
3 $x \leftarrow (m', \sigma_{\mathsf{TS}})$
4 Return x

$\underline{\mathsf{EmbInv}^{\ell,k}_{\mathsf{sig}}(x)}$:
5 $(m, \sigma_{\mathsf{TS}}) \leftarrow x$
6 $\sigma_{\mathsf{S}} \leftarrow \sigma_{\mathsf{TS}}[1..\ell]$
7 Return $((1 \,\|\, \sigma_{\mathsf{S}}), m)$

Figure 13: Fixed predicate $\mathsf{P}_{\mathsf{sig}}$ and *a* correct embedding for signatures. We assume $\ell \leq k$. The embedding space is $\mathsf{Emb}^{\ell,k}_{\mathsf{sig}}.\mathsf{ES} = \{0,1\}^{\ell+1}$; we have $z \in \{0,1\}^{\ell+1}$. In our construction we would have $\ell = \mathsf{S.sl}$ and $k = \mathsf{TS.sl}$.

Game $\mathbf{G}^{\text{mu-ufcma}}_{\mathsf{TS},n}$

Init:
1 For $i = 1, 2, \ldots, n$ do $(s_i, v_i) \leftarrow\!\!\$\ \mathsf{TS}$
2 $\mathcal{Q} \leftarrow \emptyset$
3 Return $(v_1, v_2, \ldots, v_n)$

Sign(i, m):
4 $\sigma \leftarrow\!\!\$\ s_i(m)$; $\mathcal{Q} \leftarrow \mathcal{Q} \cup \{(i, m)\}$
5 Return σ

Fin(i^*, m^*, σ^*):
6 Return $((i^*, m^*) \notin \mathcal{Q})$
$\wedge\ (v_{i^*}(m^*, \sigma^*) = 1)$

Game $\mathbf{G}^{\text{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}$

Init:
1 For $i = 1, 2, \ldots, n$ do $(s_i, v_i) \leftarrow\!\!\$\ \mathsf{TS}$
2 $((\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n), e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v_1, v_2, \ldots, v_n)$
3 $\mathcal{X} \leftarrow \emptyset$
4 Return $((v_1, v_2, \ldots, v_n), (\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n))$

ESign(i, m'):
5 $(m, \sigma) \leftarrow\!\!\$\ e(i, m', 1)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{(i, m)\}$
6 Return σ

Sign(i, m):
7 $\sigma \leftarrow\!\!\$\ s_i(m)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{(i, m)\}$
8 Return σ

Fin(i^*, m^*, σ^*):
9 Return $((i^*, m^*) \notin \mathcal{X}) \wedge (\widetilde{v}_{i^*}(m^*, \sigma^*) = 1)$

Figure 14: **Left:** The multi-user uf-cma game over n users. **Right:** Forgery-finding exclusivity (ffe) of an ASA on signature verification.

---

addition of the exploit-finding ESign oracle, following the framework of the prior two sections. If A is an adversary, we let $\mathbf{Adv}^{\text{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}(A) = \Pr\left[\mathbf{G}^{\text{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}(A)\right]$ be its ffe advantage.

Construction. We construct a multi-user ASA on signatures using a slight modification to our main transform. Given a signature scheme S, a family of verification functions V for target signature scheme TS, and an embedding function $\mathsf{Emb}_{\mathsf{sig}}$ that is compatible with S, V and is correct for predicate $\mathsf{P}_{\mathsf{sig}}$, we let $\widetilde{\mathsf{V}} = \mathbf{ASA}_n[\mathsf{V}, \mathsf{S}, \mathsf{Emb}_{\mathsf{sig}}]$. This is specified in Figure 15, both for the signature case and for the more general mu case. $\widetilde{\mathsf{V}}$ is described on the left side of Figure 15 and is constructed using V, S, and $\mathsf{Emb}_{\mathsf{sig}}$.

In the following proposition and theorem statement, we prove that this is an effective ASA, satisfying utility and ffe exclusivity.

$\widetilde{\mathsf{V}}(v_1, v_2, \ldots, v_n)$:

1 $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
2 Define as below:
  $\widetilde{v}_i : \{0,1\}^* \to \{0,1\}$
  $e : \{0,1\}^{\lceil \log n \rceil} \times \{0,1\}^* \times \{0,1\} \to \{0,1\}^*$
3 Return $((\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n), e)$

$\widetilde{v}_i(x)$:

4 $((1 \,\|\, \sigma_{\mathsf{S}}), m) \leftarrow \mathsf{EmbInv}_{\mathsf{sig}}(x)$
5 If $\mathsf{S.Vfy}(vk, (i, 1, m), \sigma_{\mathsf{S}})$ then return 1
6 Else return $v_i(x)$

$e(i, u = m', y)$:

7 Require: $y = 1$
8 $\sigma_{\mathsf{S}} \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (i, 1, m'))$
9 $x \leftarrow \mathsf{Emb}_{\mathsf{sig}}((1 \,\|\, \sigma_{\mathsf{S}}), m')$
10 Return x

$\widetilde{\mathsf{F}}(f_1, f_2, \ldots, f_n)$:

1 $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
2 Define as below:
  $\widetilde{f}_i : \{0,1\}^* \to \{0,1\}^{\mathsf{F.ol}}$
  $e : \{0,1\}^{\lceil \log n \rceil} \times \{0,1\}^* \times \{0,1\}^{\mathsf{F.ol}} \to \{0,1\}^*$
3 Return $((\widetilde{f}_1, \widetilde{f}_2, \ldots, \widetilde{f}_n), e)$

$\widetilde{f}_i(x)$:

4 $w \leftarrow \mathsf{Emb}^{-1}(x)$
5 If $(w = \perp)$ then return $f_i(x)$
6 $((y \,\|\, \sigma), u) \leftarrow w$
7 If $\mathsf{S.Vfy}(vk, (i, y, u), \sigma)$ then return y
8 Else return $f_i(x)$

$e(i, u, y)$:

9 Require: $y \in \{0,1\}^{\mathsf{F.ol}}$
10 $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (i, y, u))$
11 $x \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$
12 Return x

Figure 15: **Left:** Construction of an ASA on multi-user signature verification. **Right:** Our general ASA on public functions from Section 4, adapted to multiple users. Recall that vk is considered hardcoded in $\widetilde{v}_i$ (or $\widetilde{f}_i$) and sk is hardcoded in e.

Proposition 7.1 *Let* S *be a signature scheme,* TS *a target signature scheme with verification family* V*, and* $\mathsf{Emb}_{\mathsf{sig}}$ *an embedding function for predicate* $\mathsf{P}_{\mathsf{sig}}$ *which is compatible with* S, V*. Assume* $\mathsf{S.sl} \leq \mathsf{TS.sl}$*. Let* $\widetilde{\mathsf{V}} = \mathbf{ASA}_n[\mathsf{V}, \mathsf{S}, \mathsf{Emb}_{\mathsf{sig}}]$ *as in Figure 15. If* S *and* $\mathsf{Emb}_{\mathsf{sig}}$ *are correct, then* $\widetilde{\mathsf{V}}$ *achieves utility for* $\mathsf{P}_{\mathsf{sig}}$*.*

Proof of Proposition 7.1: For $i = 1, 2, \ldots, n$ let $(s_i, v_i) \leftarrow\!\!\$\ \mathsf{TS}$ and let $((\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n), e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v_1, v_2, \ldots, v_n)$. Let $m' \in \{0,1\}^*$ be any message and i be any user $1 \leq i \leq n$. Utility asks that the exploit algorithm can produce a signature on m' which validates for user i.

As defined in Figure 15, e produces $x \leftarrow e(i, u = m', y = 1)$ where $x = \mathsf{Emb}_{\mathsf{sig}}((1 \,\|\, \sigma_{\mathsf{S}}), m')$ and $\sigma_{\mathsf{S}} \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (i, 1, m'))$. That is, e outputs some verification input x which embeds both m' and σ_{S}. If $\mathsf{Emb}_{\mathsf{sig}}$ is a correct embedding function then $\mathsf{P}_{\mathsf{sig}}(x, u = m') = 1$. (An example of a correct embedding is in Figure 13. Recall that $\mathsf{P}_{\mathsf{sig}}$ ensures that m' is in fact the message in x.)

The second requirement of utility is that $\widetilde{v}_i(x) = 1$; that is, that a correct sigature on m' for user i is produced. This is ensured by lines 4,5 of $\widetilde{v}_i$ in Figure 15. On this input x, which is a correctly produced embedding, line 4 parses $((1 \,\|\, \sigma_{\mathsf{S}}), m) \leftarrow \mathsf{EmbInv}_{\mathsf{sig}}(x)$, where $m = m'$ by correctness of the embedding. Now line 5 computes $\mathsf{S.Vfy}(vk, (i, 1, m'), \sigma_{\mathsf{S}})$ which passes because, as in the paragraph above, σ_{S} is a signature on $(i, 1, m')$ using scheme S. Verification thus returns 1 on line 5, proving that utility is achieved. ∎

Theorem 7.2 *Let* S *be a signature scheme,* TS *a target signature scheme with verification family* V*, and* $\mathsf{Emb}_{\mathsf{sig}}$ *a correct embedding function for predicate* $\mathsf{P}_{\mathsf{sig}}$ *which is compatible with* S, V*. Assume* $\mathsf{S.sl} \leq \mathsf{TS.sl}$*. Let* $\widetilde{\mathsf{V}} = \mathbf{ASA}_n[\mathsf{V}, \mathsf{S}, \mathsf{Emb}_{\mathsf{sig}}]$ *as in Figure 15. Given an adversary* A *against the ffe*

exclusivity of $\widetilde{\mathsf{V}}$ *we can build adversaries* $A_{\mathsf{S}}, A_{\mathsf{TS}}$ *such that*

$$\mathbf{Adv}^{\mathrm{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}(A) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}}) + \mathbf{Adv}^{\text{mu-ufcma}}_{\mathsf{TS},n}(A_{\mathsf{TS}}) \ . \tag{11}$$

If A *makes* q_s, q_e SIGN, ESIGN *queries, respectively, then* A_{S} *makes* q_e SIGN *queries and* A_{TS} *makes* q_s SIGN *queries. The running times of* $A_{\mathsf{S}}, A_{\mathsf{TS}}$ *are close to that of* A*.*

We present the proof of Theorem 7.2 in Appendix B.

SINGLE-USER FFE $\not\Longrightarrow$ MULTI-USER FFE. Above, we defined game $\mathbf{G}^{\mathrm{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}$ in a multi-user setting. In contrast to the unforgeability of a normal signature scheme, ffe unforgeability in the single-user setting does not generally imply unforgeability in the multi-user setting. The intuitive reason for this is that the signature created using the exploit algorithm depends on the target user i for which the signature is forged. As a concrete counterexample, we can compare construction **ASA** to $\mathbf{ASA}_n$ as defined in Figure 15. In the latter, the exploit algorithm computes $\sigma_{\mathsf{S}} \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (i, 1, m'))$ while i is not included in the signed value in the former. In the multi-user setting, if i is not included, one could use the fact that one exploit-produced signature will verify for different (if not all) users. Multi-user considerations thus arise for this ASA setting.

8 Application: Forged certificates

CERTIFICATES AND PKI. In this section we discuss an application and embedding method which is relevant to hash functions and signatures as used in public-key infrastructure (PKI). The usual realization of PKI uses X.509 certificates and certificate authorities. For simplicity, suppose there is one honest CA who is operating with signature scheme S_{ca} and hash function $h \in \mathsf{H}_{\mathsf{ca}}$. Let $(vk_{\mathsf{ca}}, sk_{\mathsf{ca}})$ be the verification and signing keys of the CA; in our ASA syntax the signature verification algorithm is $v(\cdot) = \mathsf{S}_{\mathsf{ca}}.\mathsf{Vfy}(vk_{\mathsf{ca}}, \cdot)$. Thus all users who use the PKI with this CA have implementations of h and v (along with vk_{ca}) on their devices.

As specified in RFC 5280 [23], a certificate `C` consists of a sequence of key-value pairs. The important fields for our discussion are:

- `C.tbsCert`, consisting of the certificate's identifying, validity, and other certificate data. At a minimum, this specifies the CA who signed the certificate and includes information to recover vk_{ca}.
- `C.sigAlg`, the name of the signature algorithm, such as "PKCS #1 SHA-256 With RSA Encryption." Along with the first item above, this allows one to recover algorithms h and v.
- `C.sigValue`, a signature on message `C.tbsCert`, using the algorithm specified in `C.sigAlg` and the CA's signing key sk_{ca}.

Issuance of a certificate takes as input a $\texttt{tbsCert}'$ and auxiliary information `csr` (representing a certificate signing request) to produce either $\perp$, or a signed certificate `C`. Deterministic validation of a certificate takes as input a certificate `C` and auxiliary information `aux` (representing a certificate chain, and local store of root certificates) to produce a bit $d \in \{0, 1\}$.

In our discussion, we are more interested in validation than issuance. At a high level, validation Validate proceeds as in Figure 16. We now discuss two applications of ASAs on hash functions and signatures, though we also remark that the certificate validation process has other "general public functions" to which our general framework could apply.

ASA ON THE HASH FUNCTION. Let $\widetilde{\mathsf{H}} = \mathbf{ASA}[\mathsf{H}_{\mathsf{ca}}, \mathsf{S}, \mathsf{Emb}_{\mathsf{cert}}]$ given a signature scheme S with *short* signatures and an embedding function that will be specified shortly. Let $(\tilde{h}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{H}}(h)$. As in Figure 16, an attacker substitutes h with $\tilde{h}$ on a user's device.

$\underline{\mathsf{Validate}(\mathtt{C}, \mathtt{aux})}$:

1 First, perform expiry, revocation, and other checks.
2 Extract vk_{ca} from $(\mathtt{C}, \mathtt{aux})$
3 Extract (h, v) from $\mathtt{C.sigAlg}$ and vk_{ca}
4 $y \leftarrow h(\mathtt{C.tbsCert})$
5 Return $v(y, \mathtt{C.sigValue})$

ASA on h changes line 4:

6 $y \leftarrow \widetilde{h}(\mathtt{C.tbsCert})$

ASA on v changes line 5:

7 Return $\widetilde{v}(y, \mathtt{C.sigValue})$

Figure 16: Validation of an X.509 certificate, simplified. Recall that $v(\cdot) = \mathsf{S_{ca}.Vfy}(vk_{\mathsf{ca}}, \cdot)$.

$\underline{\mathsf{P}_{\mathsf{cert}}(x = \mathtt{tbsCert}^*, u = \mathtt{tbsCert}')}$:

1 If $((\mathtt{tbsCert}^*.\mathtt{f}_h = \bot) \vee (\mathtt{tbsCert}^*.\mathtt{f}_\sigma = \bot))$ then return false
2 If $((\mathtt{tbsCert}^*.\mathtt{f} = \mathtt{tbsCert}'.\mathtt{f})$ for all other fields $\mathtt{f})$ then return true
3 Else return false

$\underline{\mathsf{Emb}_{\mathsf{cert}}((y \,\|\, \sigma), u = \mathtt{tbsCert}')}$:

4 For all fields $\mathtt{f}$, do: $\mathtt{tbsCert}^*.\mathtt{f} \leftarrow \mathtt{tbsCert}'.\mathtt{f}$
5 $\mathtt{tbsCert}^*.\mathtt{f}_h \leftarrow y$; $\mathtt{tbsCert}^*.\mathtt{f}_\sigma \leftarrow \sigma$
6 Return $\mathtt{tbsCert}^*$

$\underline{\mathsf{Emb}^{-1}_{\mathsf{cert}}(x = \mathtt{tbsCert}^*)}$:

7 $y \leftarrow \mathtt{tbsCert}^*.\mathtt{f}_h$; $\sigma \leftarrow \mathtt{tbsCert}^*.\mathtt{f}_\sigma$
8 $\mathtt{tbsCert}'.\mathtt{f} \leftarrow \mathtt{tbsCert}^*.\mathtt{f}$ for all fields $\mathtt{f}$ except $\mathtt{f}_h, \mathtt{f}_\sigma$
9 Return $((y \,\|\, \sigma), \mathtt{tbsCert}')$

$\underline{\text{Certificate forgery using } \widetilde{\mathsf{H}}}$:

10 $(\tilde{h}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{H}}(h)$
11 Choose a target honest certificate $\mathtt{C}$ with hash y.
12 Choose any $\mathtt{tbsCert}'$ which specifies the same issuer CA and vk_{ca} as $\mathtt{C}$.
13 $\mathtt{tbsCert}^* \leftarrow\!\!\$\ e(u = \mathtt{tbsCert}', y)$
14 $\mathtt{C}^*.\mathtt{tbsCert} \leftarrow \mathtt{tbsCert}^*$
15 $\mathtt{C}^*.\mathtt{sigAlg}$ specifies (h, v)
16 $\mathtt{C}^*.\mathtt{sigValue} \leftarrow \mathtt{C.sigValue}$
17 Return $\mathtt{C}^*$

Figure 17: **Above:** Predicate $\mathsf{P}_{\mathsf{cert}}$ which captures whether data $\mathtt{tbsCert}^*$ embeds all of the data from chosen $\mathtt{tbsCert}'$, along with *a* correct embedding function for this predicate. This captures the attacker's goal of using arbitrary data in their forgery. We give an explicit example of this embedding in Appendix C. **Below:** How an ASA on a hash function h allows the attacker to forge certificates. We use $\mathtt{f}$ to denote any X.509 field name. Field labels $\mathtt{f}_h, \mathtt{f}_\sigma$ are fixed.

At a high level, certificate forgery proceeds by choosing a target hash y for which a CA signature is already known, on an honest certificate $\mathtt{C}$. Then the attacker finds a preimage certificate $\mathtt{C}^*$ which maps to y under $\widetilde{h}$. (Thus the signature on $\mathtt{C}$ may be reused.) The ability to specify what this preimage certificate "looks like" is captured by a predicate and embedding. We give one such example in Figure 17. Here, the predicate enforces that the attacker can choose any desired certificate data $\mathtt{tbsCert}'$, and can find a forgery $\mathtt{C}^*$ which differs only by adding $(\mathsf{H_{ca}.ol} + \mathsf{S.sl})$

bits. If $\widetilde{\mathsf{H}}$ is effective for this $\mathsf{P}_{\mathsf{cert}}$, and if $\mathsf{Validate}(\mathtt{C}, \mathtt{aux}) = 1$, then $\mathsf{Validate}(\mathtt{C}^*, \mathtt{aux}) = 1$. The steps are written in more detail in Figure 17. Moreover, we give an explicit example of a certificate embedding in Appendix C but we remark that our transform works for *any* correct predicate and embedding. In particular, one could design other ways to embed $(\mathsf{H}_{\mathsf{ca}}.\mathsf{ol} + \mathsf{S}.\mathsf{sl})$ bits in a certificate.

Although we consider the ASA setting, we note that a variety of work has already studied the repercussions of PKI allowing weak hash functions, in particular MD5 [48, 49, 61, 65, 69]. At Eurocrypt 2007 [60], Stevens, Lenstra, and de Weger presented two X.509 certificates with the same MD5 hash value and thus signature. The two certificates were produced at the same time, and the cost was estimated to be "2 months real time" [60]. In the ASA model, an attacker wants to forge (almost) arbitrary certifcates, at arbitrary times, and to do so easily.

ASA ON SIGNATURE VERIFICATION. Rather than attacking the hashing step of validation, one could attack the signature verification. Let $(\widetilde{v}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v)$ for an ASA $\widetilde{\mathsf{V}}$. As in Figure 16, an attacker substitutes v with $\widetilde{v}$ on a user's device. Given any certificate data $\mathtt{C}^*.\mathtt{tbsCert}$, the task is now to set $y \leftarrow h(\mathtt{C}^*.\mathtt{tbsCert})$ and find $\mathtt{C}^*.\mathtt{sigValue}$ such that $\widetilde{v}(y, \mathtt{C}^*.\mathtt{sigValue}) = 1$. Note that the embedding task is already accomplished by the utility definition for a signature ASA; the message y may be arbitrarily selected, and the exploit algorithm returns $(y, \sigma) \leftarrow\!\!\$\ e(y, 1)$ such that $\widetilde{v}(y, \sigma) = 1$. Now we simply set $\mathtt{C}^*.\mathtt{sigValue} \leftarrow \sigma$.

Thus, if we instantiate $\widetilde{\mathsf{V}} = \mathbf{ASA}[\mathsf{V}_{\mathsf{ca}}, \mathsf{S}, \mathsf{Emb}_{\mathsf{sig}}]$ with a signature scheme such that $\mathsf{S}.\mathsf{sl} \leq \mathsf{S}_{\mathsf{ca}}.\mathsf{sl}$, the results of Section 7 show that the ASA can construct arbitrary certificates with validating signatures under $\widetilde{v}$. (No special certificate embedding is needed for this ASA.)

MANY USERS AND CAS. We have only described a single-user setting in this section. In Section 7, we pointed out the difference between single-user and multi-user ffe exclusivity. This arises in practice in PKI, as many users are trusting several CAs. The observation in the signature setting was that the exploit algorithm e must consider the user i (or their identifier) when producing a preimage, which should depend on i. In our construction, this meant that i was included in the signature embedded in the found preimage. The same observation would apply in a many-user, many-CA PKI setting if the ASA aims to prevent exploit-produced preimages from being reused for different users.

References

[1] H. Abelson, R. Anderson, S. M. Bellovin, J. Benaloh, M. Blaze, W. Diffie, J. Gilmore, M. Green, S. Landau, P. G. Neumann, R. L. Rivest, J. I. Schiller, B. Schneier, M. Specter, and D. J. Weitzner. Keys under doormats: Mandating insecurity by requiring government access to all data and communications, 2015. https://dspace.mit.edu/bitstream/handle/1721.1/97690/MIT-CSAIL-TR-2015-026.pdf. (Cited on 5.)

[2] A. Albertini, J.-P. Aumasson, M. Eichlseder, F. Mendel, and M. Schläffer. Malicious hashing: Eve's variant of SHA-1. In A. Joux and A. M. Youssef, editors, *SAC 2014*, volume 8781 of *LNCS*, pages 1–19. Springer, Heidelberg, Aug. 2014. (Cited on 4, 6, 14.)

[3] M. Armour and B. Poettering. Subverting decryption in AEAD. In M. Albrecht, editor, *17th IMA International Conference on Cryptography and Coding*, volume 11929 of *LNCS*, pages 22–41. Springer, Heidelberg, Dec. 2019. (Cited on 2, 5.)

[4] G. Ateniese, D. Francati, B. Magri, and D. Venturi. Public immunization against complete subversion without random oracles. In R. H. Deng, V. Gauthier-Umaña, M. Ochoa, and

M. Yung, editors, *ACNS 19*, volume 11464 of *LNCS*, pages 465–485. Springer, Heidelberg, June 2019. (Cited on 5.)

[5] G. Ateniese, B. Magri, and D. Venturi. Subversion-resilient signature schemes. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 364–375. ACM Press, Oct. 2015. (Cited on 2, 5, 20.)

[6] B. Auerbach, M. Bellare, and E. Kiltz. Public-key encryption resistant to parameter subversion and its realization from efficiently-embeddable groups. In M. Abdalla and R. Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 348–377. Springer, Heidelberg, Mar. 2018. (Cited on 6.)

[7] F. Banfi, K. Gegier, M. Hirt, U. Maurer, and G. Rito. Anamorphic encryption, revisited. In M. Joye and G. Leander, editors, *Advances in Cryptology – EUROCRYPT 2024*, pages 3–32, Cham, 2024. Springer Nature Switzerland. (Cited on 6.)

[8] M. Bellare, G. Fuchsbauer, and A. Scafuro. NIZKs with an untrusted CRS: Security in the face of parameter subversion. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 777–804. Springer, Heidelberg, Dec. 2016. (Cited on 6, 17.)

[9] M. Bellare, J. Jaeger, and D. Kane. Mass-surveillance without the state: Strongly undetectable algorithm-substitution attacks. In I. Ray, N. Li, and C. Kruegel, editors, *ACM CCS 2015*, pages 1431–1440. ACM Press, Oct. 2015. (Cited on 2, 5.)

[10] M. Bellare, K. G. Paterson, and P. Rogaway. Security of symmetric encryption against mass surveillance. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part I*, volume 8616 of *LNCS*, pages 1–19. Springer, Heidelberg, Aug. 2014. (Cited on 2, 3, 5.)

[11] M. Bellare and P. Rogaway. The security of triple encryption and a framework for code-based game-playing proofs. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 409–426. Springer, Heidelberg, May / June 2006. (Cited on 6, 9, 13, 15, 32.)

[12] P. Bemmann, S. Berndt, and R. Chen. Subversion-resilient signatures without random oracles. In C. Pöpper and L. Batina, editors, *Applied Cryptography and Network Security*, pages 351–375, Cham, 2024. Springer Nature Switzerland. (Cited on 14.)

[13] P. Bemmann, R. Chen, and T. Jager. Subversion-resilient public key encryption with practical watchdogs. In J. Garay, editor, *PKC 2021, Part I*, volume 12710 of *LNCS*, pages 627–658. Springer, Heidelberg, May 2021. (Cited on 5.)

[14] S. Berndt, J. Wichelmann, C. Pott, T.-H. Traving, and T. Eisenbarth. ASAP: Algorithm substitution attacks on cryptographic protocols. In Y. Suga, K. Sakurai, X. Ding, and K. Sako, editors, *ASIACCS 22*, pages 712–726. ACM Press, May / June 2022. (Cited on 2, 5.)

[15] D. J. Bernstein, T. Lange, and R. Niederhagen. Dual EC: A standardized back door. In P. Ryan, D. Naccache, and J.-J. Quisquater, editors, *The New Codebreakers*. Springer, Heidelberg, 2016. (Cited on 5.)

[16] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the Weil pairing. In C. Boyd, editor, *ASIACRYPT 2001*, volume 2248 of *LNCS*, pages 514–532. Springer, Heidelberg, Dec. 2001. (Cited on 34.)

[17] D. Boneh, E. Shen, and B. Waters. Strongly unforgeable signatures based on computational Diffie-Hellman. In M. Yung, Y. Dodis, A. Kiayias, and T. Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 229–240. Springer, Heidelberg, Apr. 2006. (Cited on 7.)

[18] S. Chakraborty, C. Ganesh, and P. Sarkar. Reverse firewalls for oblivious transfer extension and applications to zero-knowledge. In C. Hazay and M. Stam, editors, *EUROCRYPT 2023, Part I*, volume 14004 of *LNCS*, pages 239–270. Springer, Heidelberg, Apr. 2023. (Cited on 17.)

[19] R. Chen, X. Huang, and M. Yung. Subvert KEM to break DEM: Practical algorithm-substitution attacks on public-key encryption. In S. Moriai and H. Wang, editors, *ASIACRYPT 2020, Part II*, volume 12492 of *LNCS*, pages 98–128. Springer, Heidelberg, Dec. 2020. (Cited on 2, 5.)

[20] S. S. M. Chow, A. Russell, Q. Tang, M. Yung, Y. Zhao, and H.-S. Zhou. Let a non-barking watchdog bite: Cliptographic signatures with an offline watchdog. In D. Lin and K. Sako, editors, *PKC 2019, Part I*, volume 11442 of *LNCS*, pages 221–251. Springer, Heidelberg, Apr. 2019. (Cited on 2.)

[21] B. Cogliati, J. Ethan, and A. Jha. Subverting telegram's end-to-end encryption. IACR *Transactions on Symmetric Cryptology*, 2023. https://tosc.iacr.org/index.php/ToSC/article/view/10302/9747. (Cited on 2, 5.)

[22] S. Contini, A. K. Lenstra, and R. Steinfeld. VSH, an efficient and provable collision-resistant hash function. In S. Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 165–182. Springer, Heidelberg, May / June 2006. (Cited on 4, 14.)

[23] D. Cooper, S. Santesson, S. Farrell, S. Boeyen, R. Housley, and W. Polk. RFC 5280: Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile, May 2008. (Cited on 23.)

[24] R. Cramer and V. Shoup. Signature schemes based on the strong RSA assumption. In J. Motiwalla and G. Tsudik, editors, *ACM CCS 99*, pages 46–51. ACM Press, Nov. 1999. (Cited on 7.)

[25] C. Crépeau and A. Slakmon. Simple backdoors for RSA key generation. In M. Joye, editor, *CT-RSA 2003*, volume 2612 of *LNCS*, pages 403–416. Springer, Heidelberg, Apr. 2003. (Cited on 20.)

[26] J. P. Degabriele, P. Farshim, and B. Poettering. A more cautious approach to security against mass surveillance. In G. Leander, editor, *FSE 2015*, volume 9054 of *LNCS*, pages 579–598. Springer, Heidelberg, Mar. 2015. (Cited on 2, 5.)

[27] J. P. Degabriele, K. G. Paterson, J. C. N. Schuldt, and J. Woodage. Backdoors in pseudorandom number generators: Possibility and impossibility results. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 403–432. Springer, Heidelberg, Aug. 2016. (Cited on 6.)

[28] D. Derler, K. Samelin, and D. Slamanig. Bringing order to chaos: The case of collision-resistant chameleon-hashes. In A. Kiayias, M. Kohlweiss, P. Wallden, and V. Zikas, editors, *PKC 2020, Part I*, volume 12110 of *LNCS*, pages 462–492. Springer, Heidelberg, May 2020. (Cited on 14.)

[29] Y. Dodis, C. Ganesh, A. Golovnev, A. Juels, and T. Ristenpart. A formal treatment of backdoored pseudorandom generators. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 101–126. Springer, Heidelberg, Apr. 2015. (Cited on 6.)

[30] Y. Dodis, I. Mironov, and N. Stephens-Davidowitz. Message transmission with reverse firewalls—secure communication on corrupted machines. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part I*, volume 9814 of *LNCS*, pages 341–372. Springer, Heidelberg, Aug. 2016. (Cited on 5.)

[31] N. Döttling, S. Garg, Y. Ishai, G. Malavolta, T. Mour, and R. Ostrovsky. Trapdoor hash functions and their applications. In A. Boldyreva and D. Micciancio, editors, *CRYPTO 2019, Part III*, volume 11694 of *LNCS*, pages 3–32. Springer, Heidelberg, Aug. 2019. (Cited on 14.)

[32] E. Felten. The linux backdoor attempt of 2003, 2013. https://freedom-to-tinker.com/2013/10/09/the-linux-backdoor-attempt-of-2003/. (Cited on 5.)

[33] M. Fischlin and F. Günther. Verifiable verification in cryptographic protocols. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, CCS '23, page 3239–3253, New York, NY, USA, 2023. Association for Computing Machinery. (Cited on 17.)

[34] M. Fischlin, C. Janson, and S. Mazaheri. Backdoored hash functions: Immunizing HMAC and HKDF. In S. Chong and S. Delaune, editors, *CSF 2018 Computer Security Foundations Symposium*, pages 105–118. IEEE Computer Society Press, 2018. (Cited on 2, 4, 6, 10, 14.)

[35] A. Freund. Openwall oss-security mailing list, 29 Mar. 2024. https://www.openwall.com/lists/oss-security/2024/03/29/4. (Cited on 6.)

[36] G. Fuchsbauer. Subversion-zero-knowledge SNARKs. In M. Abdalla and R. Dahab, editors, *PKC 2018, Part I*, volume 10769 of *LNCS*, pages 315–347. Springer, Heidelberg, Mar. 2018. (Cited on 17.)

[37] R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 123–139. Springer, Heidelberg, May 1999. (Cited on 7.)

[38] E.-J. Goh, D. Boneh, B. Pinkas, and P. Golle. The design and implementation of protocol-based hidden key recovery. In C. Boyd and W. Mao, editors, *ISC 2003*, volume 2851 of *LNCS*, pages 165–179. Springer, Heidelberg, Oct. 2003. (Cited on 2, 5.)

[39] O. Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004. (Cited on 7.)

[40] S. Goldwasser, M. P. Kim, V. Vaikuntanathan, and O. Zamir. Planting undetectable backdoors in machine learning models : [extended abstract]. In *63rd FOCS*, pages 931–942. IEEE Computer Society Press, Oct. / Nov. 2022. (Cited on 2, 3, 4, 6.)

[41] P. Hodges and D. Stebila. Algorithm substitution attacks: State reset detection and asymmetric modifications. *IACR Trans. Symm. Cryptol.*, 2021(2):389–422, 2021. (Cited on 2.)

[42] D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. *Journal of Cryptology*, 25(3):484–527, July 2012. (Cited on 14.)

[43] T. Horel, S. Park, S. Richelson, and V. Vaikuntanathan. How to subvert backdoored encryption: Security against adversaries that decrypt all ciphertexts. In A. Blum, editor, *ITCS 2019*, volume 124, pages 42:1–42:20. LIPIcs, Jan. 2019. (Cited on 6.)

[44] A. Joux, J. Loss, and B. Wagner. Kleptographic attacks against implicit rejection. Cryptology ePrint Archive, Report 2024/260, 2024. https://eprint.iacr.org/2024/260. (Cited on 2, 5.)

[45] B. Kaliski. PKCS #5: Password-based cryptography specification. RSA Laboratories, Sept. 2000. Version 2.0. (Cited on 17.)

[46] H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS 2000*. The Internet Society, Feb. 2000. (Cited on 14.)

[47] M. Kutylowski, G. Persiano, D. H. Phan, M. Yung, and M. Zawada. Anamorphic signatures: Secrecy from a dictator who only permits authentication! In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part II*, volume 14082 of *LNCS*, pages 759–790. Springer, Heidelberg, Aug. 2023. (Cited on 6.)

[48] A. Lenstra, X. Wang, and B. de Weger. Colliding x.509 certificates. Cryptology ePrint Archive, Report 2005/067, 2005. https://eprint.iacr.org/2005/067. (Cited on 25.)

[49] A. K. Lenstra and B. de Weger. On the possibility of constructing meaningful hash collisions for public keys. In C. Boyd and J. M. G. Nieto, editors, *ACISP 05*, volume 3574 of *LNCS*, pages 267–279. Springer, Heidelberg, July 2005. (Cited on 25.)

[50] I. Mironov and N. Stephens-Davidowitz. Cryptographic reverse firewalls. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 657–686. Springer, Heidelberg, Apr. 2015. (Cited on 5.)

[51] OpenSSL. https://www.openssl.org/. (Cited on 34.)

[52] G. Persiano, D. H. Phan, and M. Yung. Anamorphic encryption: Private communication against a dictator. In O. Dunkelman and S. Dziembowski, editors, *EUROCRYPT 2022, Part II*, volume 13276 of *LNCS*, pages 34–63. Springer, Heidelberg, May / June 2022. (Cited on 6.)

[53] P. Ravi, S. Bhasin, A. Chattopadhyay, Aikata, and S. S. Roy. Backdooring post-quantum cryptography: Kleptographic attacks on lattice-based KEMs. Cryptology ePrint Archive, Report 2022/1681, 2022. https://eprint.iacr.org/2022/1681. (Cited on 2, 5.)

[54] A. Russell, Q. Tang, M. Yung, and H.-S. Zhou. Cliptography: Clipping the power of kleptographic attacks. In J. H. Cheon and T. Takagi, editors, *ASIACRYPT 2016, Part II*, volume 10032 of *LNCS*, pages 34–64. Springer, Heidelberg, Dec. 2016. (Cited on 2, 5.)

[55] A. Russell, Q. Tang, M. Yung, and H.-S. Zhou. Generic semantic security against a kleptographic adversary. In B. M. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *ACM CCS 2017*, pages 907–922. ACM Press, Oct. / Nov. 2017. (Cited on 2, 5.)

[56] A. Russell, Q. Tang, M. Yung, and H.-S. Zhou. Correcting subverted random oracles. In H. Shacham and A. Boldyreva, editors, *CRYPTO 2018, Part II*, volume 10992 of *LNCS*, pages 241–271. Springer, Heidelberg, Aug. 2018. (Cited on 14.)

[57] D. Shumow and N. Ferguson. On the possibility of a back door in the NIST sp800-90 dual EC PRNG. *CRYPTO'07* Rump Session, 2007. https://rump2007.cr.yp.to/15-shumow.pdf. (Cited on 5.)

[58] G. J. Simmons. The prisoners' problem and the subliminal channel. In D. Chaum, editor, *CRYPTO'83*, pages 51–67. Plenum Press, New York, USA, 1983. (Cited on 6.)

[59] G. J. Simmons. The subliminal channel and digital signature. In T. Beth, N. Cot, and I. Ingemarsson, editors, *EUROCRYPT'84*, volume 209 of *LNCS*, pages 364–378. Springer, Heidelberg, Apr. 1985. (Cited on 6.)

[60] M. Stevens, A. K. Lenstra, and B. de Weger. Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 1–22. Springer, Heidelberg, May 2007. (Cited on 4, 25.)

[61] M. Stevens, A. Sotirov, J. Appelbaum, A. K. Lenstra, D. Molnar, D. A. Osvik, and B. de Weger. Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 55–69. Springer, Heidelberg, Aug. 2009. (Cited on 4, 25.)

[62] T. Tiemann, S. Berndt, T. Eisenbarth, and M. Liskiewicz. "Act natural!": Having a private chat on a public blockchain. Cryptology ePrint Archive, Report 2021/1073, 2021. https://eprint.iacr.org/2021/1073. (Cited on 2, 5, 20.)

[63] P. R. Tiwari and M. Green. Subverting cryptographic hardware used in blockchain consensus. In J. Clark and E. Shi, editors, *FC 2024*, 2024. (Cited on 2, 5, 17.)

[64] F. Valsorda. Bluesky post, 30 Mar. 2024. https://bsky.app/profile/did:plc:x2nsupeeo52oznrmplwapppl/post/3kowjkx2njy2b. (Cited on 6.)

[65] X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35. Springer, Heidelberg, May 2005. (Cited on 25.)

[66] A. Young and M. Yung. The dark side of "black-box" cryptography, or: Should we trust capstone? In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 89–103. Springer, Heidelberg, Aug. 1996. (Cited on 2, 5.)

[67] A. Young and M. Yung. Kleptography: Using cryptography against cryptography. In W. Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 62–74. Springer, Heidelberg, May 1997. (Cited on 2, 5.)

[68] A. Young and M. Yung. The prevalence of kleptographic attacks on discrete-log based cryptosystems. In B. S. Kaliski Jr., editor, *CRYPTO'97*, volume 1294 of *LNCS*, pages 264–276. Springer, Heidelberg, Aug. 1997. (Cited on 5.)

[69] G. Zaverucha and D. Shumow. Are certificate thumbprints unique? Cryptology ePrint Archive, Report 2019/130, 2019. https://eprint.iacr.org/2019/130. (Cited on 4, 25.)

A Proof of Theorem 6.2

Proof of Theorem 6.2: Consider game G_0 of Figure 18. We rewrite the winning condition by splitting it into three checks. Similar to the proof of Theorem 5.2, we already include line 11 which sets a flag bad. We have

$$\mathbf{Adv}^{\mathrm{pfe}}_{\mathsf{PS},\widetilde{\mathsf{V}}}(A) = \Pr\left[\,\mathrm{G}_0(A)\,\right].$$

Games G_0, $\boxed{\mathrm{G}_1}$

INIT:
1 $(p, v) \leftarrow\!\!\$\ \mathsf{PS}$; $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
2 Define $\widetilde{v}$ as in Fig. 11 (using vk selected above)
3 $\mathcal{X} \leftarrow \emptyset$
4 Return $(p, v, \widetilde{v})$

SIM(ϕ'):
5 $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (1, \phi'))$
6 $x \leftarrow \mathsf{Emb}_{\mathsf{PS}}((1 \,\|\, \sigma), \phi')$
7 $\mathcal{X} \leftarrow \mathcal{X} \cup \{\phi'\}$
8 Return x

FIN(ϕ^*, π^*):
9 If $\phi^* \in \mathcal{X}$ then return false
10 If $\phi^* \in L_R$ then return false
11 If $(v(\phi^*, \pi^*) = 0 \wedge \widetilde{v}(\phi^*, \pi^*) = 1)$ then bad $\leftarrow$ true ; $\boxed{\text{return false}}$
12 Return $v(\phi^*, \pi^*)$

Figure 18: Games $\mathrm{G}_0, \mathrm{G}_1$ for the proof of Theorem 6.2. G_1 contains the boxed code and G_0 does not. Oracle SIM is as in Figure 12.

Adversary $A_{\mathsf{S}}(vk)$:
1 $(p, v) \leftarrow\!\!\$\ \mathsf{PS}$
2 Define $\widetilde{v}$ as in Fig. 11 (using the input vk)
3 $\mathcal{X} \leftarrow \emptyset$; $\mathcal{Q}_{\mathsf{S}} \leftarrow \emptyset$
4 $(\phi^*, \pi^*) \leftarrow A[\mathrm{SIM}_{\mathsf{S}}](p, v, \widetilde{v})$
5 $x^* \leftarrow (\phi^*, \pi^*)$
6 $((1 \,\|\, \sigma), \phi) \leftarrow \mathsf{EmbInv}_{\mathsf{PS}}(x^*)$
7 Return $(m = (1, \phi), \sigma)$

Oracle $\mathrm{SIM}_{\mathsf{S}}(\phi')$:
8 $\sigma \leftarrow\!\!\$\ \mathrm{SIGN}((1, \phi'))$
9 $x \leftarrow \mathsf{Emb}_{\mathsf{PS}}((1 \,\|\, \sigma_{\mathsf{S}}), \phi')$
10 $\mathcal{X} \leftarrow \mathcal{X} \cup \{\phi'\}$
11 $\mathcal{Q}_{\mathsf{S}} \leftarrow \mathcal{Q}_{\mathsf{S}} \cup \{((1, \phi'), \sigma)\}$
12 Return x

Adversary $A_{\mathsf{PS}}(p, v)$:
1 $(\widetilde{v}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{V}}(v)$; $\mathcal{X} \leftarrow \emptyset$
2 $(\phi^*, \pi^*) \leftarrow A[\mathrm{SIM}_{\mathsf{PS}}](p, v, \widetilde{v})$
3 Return (ϕ^*, π^*)

Oracle $\mathrm{SIM}_{\mathsf{PS}}(\phi')$:
4 $(\phi, \pi) \leftarrow\!\!\$\ e(\phi', 1)$; $\mathcal{X} \leftarrow \mathcal{X} \cup \{\phi'\}$
5 Return π

Figure 19: Adversaries A_{S} (left) and A_{PS} (right) for the proof of Theorem 6.2.

We next turn to game G_1 which outputs false whenever bad is set; that is, when $\widetilde{v}(\phi^*, \pi^*)$ is true, but $v(\phi^*, \pi^*)$ is not. Therefore, games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad and we have

$$\begin{aligned}\Pr[\,\mathrm{G}_0(A)\,] &= \Pr[\,\mathrm{G}_1(A)\,] + (\Pr[\,\mathrm{G}_0(A)\,] - \Pr[\,\mathrm{G}_1(A)\,]) \\ &\le \Pr[\,\mathrm{G}_1(A)\,] + \Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,]\,.\end{aligned}$$

We now construct adversaries $A_{\mathsf{S}}, A_{\mathsf{PS}}$ such that the following two equations hold:

$$\Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,] \le \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}}) \tag{12}$$

$$\Pr[\,\mathrm{G}_1(A)\,] \le \mathbf{Adv}^{\text{snd}}_{\mathsf{PS}}(A_{\mathsf{PS}})\,, \tag{13}$$

which will complete the proof of Eq. (10) in the theorem statement.

We begin with A_{S} which is in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}$ and operates according to the description in Figure 19. A_{S} runs A, responding to oracle queries via $\textsc{Sim}_{\mathsf{S}}$. These follow the responses in game G_1, with A_{S} running its own $\textsc{Sign}$ oracle on line 8. These queries to its own oracle are tracked as message-signature pairs in set $\mathcal{Q}_{\mathsf{S}}$. Now suppose that bad is set in G_1.

Then $\phi^* \notin \mathcal{X}$, $v(\phi^*, \pi^*) = 0$ and $\widetilde{v}(\phi^*, \pi^*) = 1$. In particular, $\phi^* \notin \mathcal{X}$ implies that $(1, \phi^*)$ is not in any message-signature pair in set $\mathcal{Q}_{\mathsf{S}}$ (else both would have been added on lines 10-11 of Figure 19). Moreover, since $\widetilde{v}(\phi^*, \pi^*) = 1$ the parsed signature σ is a valid signature such that $\mathsf{S.Vfy}(vk, (1, \phi^*), \sigma)$. A_{S} thus outputs this winning pair $((1, m^*), \sigma)$ in its suf-cma game. This justifies Eq. (12). Further, if A issues q queries to its simulation oracle $\textsc{Sim}_{\mathsf{S}}$, then A_{S} also issues q queries to $\textsc{Sign}$.

We next turn to adversary A_{PS} which is in game $\mathbf{G}^{\text{snd}}_{\mathsf{PS}}$ and is depicted on the right side of Figure 19. A's view is that of game G_1; initialization and $\textsc{Sim}_{\mathsf{PS}}$ return the same responses as in G_1. Now, if $\mathrm{G}_1(A)$ returns true, and since the boxed code is executed in G_1, then it must be that the same conditions are satisfied as in $\mathbf{G}^{\text{snd}}_{\mathsf{PS}}$ and A returns a winning output which proves Eq. (13).

We conclude the proof by observing that A_{S} and A_{PS} maintain running times close to that of A.∎

B Proof of Theorem 7.2

Proof of Theorem 7.2: Consider game G_0 of Figure 20. We claim that G_0 is equivalent to game $\mathbf{G}^{\text{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}$ when instantiated with $\widetilde{\mathsf{V}} = \mathbf{ASA}_n[\mathsf{V}, \mathsf{S}, \mathsf{Emb}_{\mathsf{sig}}]$. Note that the $\textsc{Init}$ and $\textsc{Sign}$ oracles are running the same steps, only with additional accounting. The $\textsc{ESign}$ oracle in G_0 has replaced the computation $e(i, m', 1)$ with its execution as per the definition of $\widetilde{\mathsf{V}}$, and the $\textsc{Fin}$ oracle of G_0 returns true if $(i^*, m^*) \notin \mathcal{X}$ and if (including the boxed code) either of $\mathsf{S.Vfy}(vk, (i^*, 1, m^*), \sigma_{\mathsf{S}}) = 1$ or $v_{i^*}(m^*, \sigma^*) = 1$. Recall that $\textsc{Fin}$ of $\mathbf{G}^{\text{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}$ returns true if $(i^*, m^*) \notin \mathcal{X}$ and $\widetilde{v}_{i^*}(m^*, \sigma^*) = 1$. The definition of $\widetilde{v}_i$ shows that $\widetilde{v}_{i^*}(m^*, \sigma^*) = 1$ precisely when either $\mathsf{S.Vfy}(vk, (i^*, 1, m^*), \sigma_{\mathsf{S}}) = 1$ or $v_{i^*}(m^*, \sigma^*) = 1$. This completes the justification that

$$\mathbf{Adv}^{\text{ffe}}_{\mathsf{TS},\widetilde{\mathsf{V}},n}(A) = \Pr[\,\mathrm{G}_0(A)\,] \,.$$

Since games $\mathrm{G}_0, \mathrm{G}_1$ are identical-until-bad, the Fundamental Lemma of Game Playing [11] implies

$$\begin{aligned}\Pr[\,\mathrm{G}_0(A)\,] &= \Pr[\,\mathrm{G}_1(A)\,] + (\Pr[\,\mathrm{G}_0(A)\,] - \Pr[\,\mathrm{G}_1(A)\,]) \\ &\le \Pr[\,\mathrm{G}_1(A)\,] + \Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,] \,.\end{aligned}$$

We now construct adversaries $A_{\mathsf{S}}, A_{\mathsf{TS}}$ such that the following two equations hold:

$$\Pr[\,\mathrm{G}_1(A)\,] \le \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}}) \tag{14}$$

$$\Pr[\,\mathrm{G}_1(A) \text{ sets } \mathsf{bad}\,] \le \mathbf{Adv}^{\text{mu-ufcma}}_{\mathsf{TS},n}(A_{\mathsf{TS}}) \,, \tag{15}$$

which will complete the proof of Eq. (11) in the theorem statement.

We begin with A_{S} which is in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}$ and operates according to the description in Figure 21. A_{S} runs A, responding to oracle queries via $\textsc{ESign}_{\mathsf{S}}$ and $\textsc{Sign}_{\mathsf{S}}$. These follow the responses in game G_1, with A_{S} running its own $\textsc{Sign}$ oracle on line 8 in responses to $\textsc{ESign}_{\mathsf{S}}$ queries. These queries to its own oracle are tracked as message-signature pairs in set $\mathcal{Q}_{\mathsf{S}}$. Now suppose that $\mathrm{G}_1(A)$ returns

Games $\boxed{\mathrm{G}_0}$, G_1

INIT:
1 For $i = 1, 2, \ldots, n$ do $(s_i, v_i) \leftarrow\!\!\$\ \mathsf{TS}$
2 $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
3 Define $(\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n)$ as in Fig. 15 (using vk selected above)
4 $\mathcal{X} \leftarrow \emptyset$; $\mathcal{Q}_{\mathsf{S}} \leftarrow \emptyset$
5 Return $((v_1, v_2, \ldots, v_n), (\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n))$

ESIGN(i, m'):
6 $\sigma_{\mathsf{S}} \leftarrow\!\!\$\ \mathsf{S.Sign}(sk, (i, 1, m'))$; $x \leftarrow \mathsf{Emb}_{\mathsf{sig}}((1 \,\|\, \sigma_{\mathsf{S}}), m')$
7 $\mathcal{X} \leftarrow \mathcal{X} \cup \{(i, m')\}$; $\mathcal{Q}_{\mathsf{S}} \leftarrow \mathcal{Q}_{\mathsf{S}} \cup \{((i, 1, m'), \sigma_{\mathsf{S}})\}$
8 Return x

SIGN(i, m):
9 $\sigma_{\mathsf{TS}} \leftarrow\!\!\$\ s_i(m)$
10 $\mathcal{X} \leftarrow \mathcal{X} \cup \{(i, m)\}$
11 Return σ_{TS}

FIN(i^*, m^*, σ^*):
12 If $((i^*, m^*) \in \mathcal{X})$ then return false
13 $x^* \leftarrow (m^*, \sigma^*)$
14 $((1 \,\|\, \sigma_{\mathsf{S}}), m) \leftarrow \mathsf{EmbInv}_{\mathsf{sig}}(x^*)$
15 $r_1 \leftarrow \mathsf{S.Vfy}(vk, (i^*, 1, m^*), \sigma_{\mathsf{S}})$
16 $r_2 \leftarrow v_{i^*}(m^*, \sigma^*)$
17 If $(r_1 = 0 \wedge r_2 = 1)$ then $\mathsf{bad} \leftarrow \mathsf{true}$; $\boxed{\mathsf{return\ true}}$
18 Return $(r_1 = 1)$

Figure 20: Games $\mathrm{G}_0, \mathrm{G}_1$ for the proof of Theorem 7.2. G_0 contains the boxed code and G_1 does not.

Adversary $A_{\mathsf{S}}(vk)$:
1 For $i = 1, 2, \ldots, n$ do $s_i, v_i) \leftarrow\!\!\$\ \mathsf{TS}$
2 Define $(\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n)$ as in Fig. 15 (using the provided input vk)
3 $\mathcal{X} \leftarrow \emptyset$; $\mathcal{Q}_{\mathsf{S}} \leftarrow \emptyset$
4 $(i^*, m^*, \sigma^*) \leftarrow A[\mathrm{ESIGN}_{\mathsf{S}}, \mathrm{SIGN}_{\mathsf{S}}]((v_1, v_2, \ldots, v_n), (\widetilde{v}_1, \widetilde{v}_2, \ldots, \widetilde{v}_n))$
5 $x^* \leftarrow (m^*, \sigma^*)$
6 $((1 \,\|\, \sigma_{\mathsf{S}}), m) \leftarrow \mathsf{EmbInv}_{\mathsf{sig}}(x^*)$
7 Return $((i^*, 1, m^*), \sigma_{\mathsf{S}})$

Oracle $\mathrm{ESIGN}_{\mathsf{S}}(i, m')$:
8 $\sigma_{\mathsf{S}} \leftarrow\!\!\$\ \mathrm{SIGN}((i, 1, m'))$; $x \leftarrow \mathsf{Emb}_{\mathsf{sig}}((1 \,\|\, \sigma_{\mathsf{S}}), m')$
9 $\mathcal{X} \leftarrow \mathcal{X} \cup \{(i, m')\}$; $\mathcal{Q}_{\mathsf{S}} \leftarrow \mathcal{Q}_{\mathsf{S}} \cup \{((i, 1, m'), \sigma_{\mathsf{S}})\}$
10 Return x

Oracle $\mathrm{SIGN}_{\mathsf{S}}(i, m)$:
11 $\sigma_{\mathsf{TS}} \leftarrow\!\!\$\ s_i(m)$
12 $\mathcal{X} \leftarrow \mathcal{X} \cup \{(i, m)\}$
13 Return σ_{TS}

Figure 21: Adversary A_{S} for the proof of Theorem 7.2.

---

true. Then $(i^*, m^*) \notin \mathcal{X}$ and $r_1 = 1$. In particular, $(i^*, m^*) \notin \mathcal{X}$ implies that $(i^*, 1, m^*)$ is not in any message-signature pair in set $\mathcal{Q}_{\mathsf{S}}$ (else both would have been added on line 9 of Figure 21). Moreover,

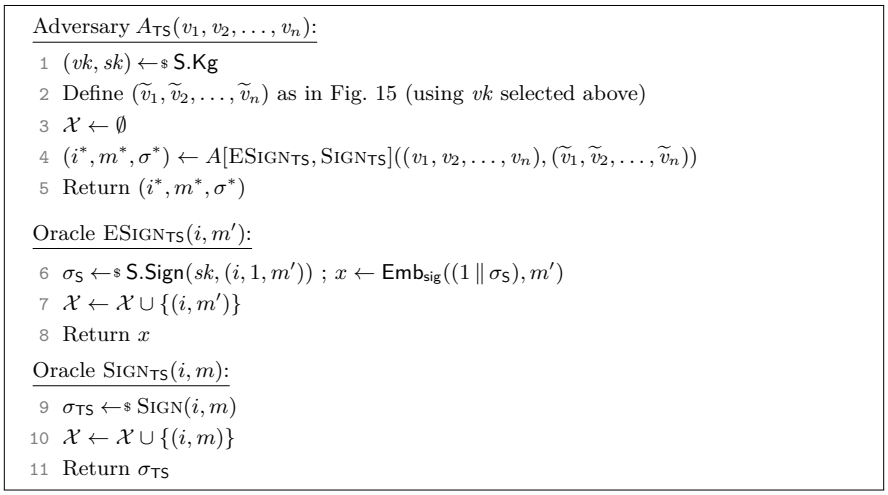

Figure 22: Adversary A_{TS} for the proof of Theorem 7.2.

since $r_1 = 1$ the parsed signature σ_{S} is a valid signature such that $\mathsf{S.Vfy}(vk, (i^*, 1, m^*), \sigma_{\mathsf{S}})$. A_{S} thus outputs this winning pair $((i^*, 1, m^*), \sigma_{\mathsf{S}})$ in its suf-cma game. This justifies Eq. (14).

Next we turn to adversary A_{TS} which is in game $\mathbf{G}^{\text{mu-ufcma}}_{\mathsf{TS},n}$ and is depicted in Figure 22. A_{TS} runs A, responding to oracle queries according to $\text{ESIGN}_{\mathsf{TS}}$ and $\text{SIGN}_{\mathsf{TS}}$. Oracle $\text{ESIGN}_{\mathsf{TS}}$ is the same as in game G_1 while oracle $\text{SIGN}_{\mathsf{TS}}$ calls A_{TS}'s own SIGN oracle to match G_1. We claim that if the flag bad is set during this execution (as per line 17 of G_1) then A_{TS} returns a valid forgery in game $\mathbf{G}^{\text{mu-ufcma}}_{\mathsf{TS},n}$. If bad is set then we have $(i^*, m^*) \notin \mathcal{X}$, $r_1 = 0$ and $r_2 = 1$. The latter means that $v_{i^*}(m^*, \sigma^*)$ and thus (i^*, m^*, σ^*) is a winning forgery for A_{TS}, proving Eq. (15).

We complete the proof of the theorem statement by noting that A_{S} makes q_e SIGN queries while A_{TS} makes q_s SIGN queries, as needed. ∎

C Forged certificate embedding

In Section 8 we gave certificate forgery as an example of a realistic embedding scenario. An ASA could be mounted either on the hash function or on the signature verification steps of certificate validation. Recall that effectiveness of an ASA on a hash function is defined relative to a predicate $\mathsf{P}_{\mathsf{cert}}$, for which our construction assumes a correct embedding function. This predicate is defined in pseudocode in Figure 17 but we here give a concrete example. Moreover, effectiveness of an ASA on signature verification already allows for *any* target message to be signed, as we cover in Section 7. Nonetheless, feasibility of an attack where forgeries "look innocuous" may ask for more sophisticated embeddings, so we provide this illustration. We focus on the hash function case.

Suppose we have selected target hash $y =$ `680f8b1123be39f4451430d6267a8159033034403ce0df1abdf11c105031d719`. This corresponds to a public certificate `C` with a valid signature `C.sigValue` where `C.sigAlg` specifies "PKCS #1 SHA-256 With RSA Encryption." The aim is now to construct $\texttt{C}^* \neq \texttt{C}$ with the same hash y; thus the signature on `C` can be reused.

An ASA attacker, following the specification of Section 8, does the following. Suppose they intend to use $\widetilde{\mathsf{H}} = \mathbf{ASA}[\mathsf{H}_{\mathsf{ca}}, \mathsf{S}, \mathsf{Emb}_{\mathsf{cert}}]$ where H_{ca} is $\mathsf{SHA256}$, S is ECDSA over `secp256k1`, and the embedding is as in Figure 17 of Section 8. We chose ECDSA because it is easily used in OpenSSL [51]; however a shorter signature scheme like BLS [16] may be easier to embed.

Certificate $\mathtt{C}^*$:

```
1  Data:
2     Version: 3 (0x2)
3     Serial Number: ...
4     Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption
5     Issuer: C.Issuer
6     Validity
7        Not Before: Jan 1 08:00:00 2024 GMT
8        Not After : Jun 1 08:00:00 2024 GMT
9     Subject: O = Big Brother, CN = *.bigbrother.com
10    Subject Public Key Info: ...
11    X509v3 extensions:
12       X509v3 Basic Constraints: critical
13          CA:TRUE
14       f_h: 680f8b1123be39f4451430d6267a8159033034403ce0df1abdf11c105031d719
15       f_σ: 304502202b978f95a853dfa2d2574ff9...56ff5dbdeed8948eb7570089e12d5
16 Signature Algorithm: PKCS #1 SHA-256 With RSA Encryption
17 C.sigValue
```

Figure 23: A certificate forgery for an ASA on $\mathsf{SHA256}$; we want $\tilde{h}(\mathtt{C}^*) = \tilde{h}(\mathtt{C}) = \mathsf{SHA256}(\mathtt{C})$ so that a signature on $\mathtt{C}$ can be reused for forgery $\mathtt{C}^*$. The highlighted lines are the overhead in constructing the hash collision; that is, these are determined by $\mathtt{C}$ or cannot be arbitrarily chosen. The remainder of the certificate, and in particular the "..." sections may be arbitrarily set by the ASA attacker. On a user's device, $\mathsf{SHA256}$ is replaced by $\tilde{h}$.

The forgery proceeds as follows. First, the substitution is generated via $(\tilde{h}, e) \leftarrow\!\!\$\ \widetilde{\mathsf{H}}(h)$, which in particular means generating (vk^*, sk^*) for ECDSA. (Recall that vk^* is hardcoded in $\tilde{h}$ while sk^* is hardcoded in e.) We select:

$$vk^* = \mathtt{04d0722759460447f1719ac66a1734054651f7c557a96166583d686}$$
$$\mathtt{ad405ca9b6f5fe47a7e425a8722edfa13be606fcbe4053ecacb27f2}$$
$$\mathtt{b0bc3dd1e83152c9a8a3}\ .$$

Next $\mathtt{tbsCert}'$ is chosen, which is the certificate data to be contained in the forgery. Section 8 discussed arbitrary data and in the Introduction, we discussed impersonation of a legitimate website by swapping out the public key in their certificate. For this example, we suppose that the attacker is aiming to forge arbitrary $\mathtt{tbsCert}'$. Now they use e to find a preimage $\mathtt{tbsCert}^*$ of target hash y, where the constraint is that $\mathtt{tbsCert}^*$ is "close to" $\mathtt{tbsCert}'$. Concretely, $\mathtt{tbsCert}^*$ adds two additional fields. In the first, $\mathtt{tbsCert}^*.\mathtt{f}_h = y$. In the second, $\mathtt{tbsCert}^*.\mathtt{f}_\sigma = \sigma$, where $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk^*, (y, \mathtt{tbsCert}'))$. For our chosen data and y, we find

$$\sigma = \mathtt{304502202b978f95a853dfa2d2574ff9980a4351e7d6c9c4fcc0529}$$
$$\mathtt{d636c750fdf4c16a8022100efbb50c105df2a4766cfa94910d3a190}$$
$$\mathtt{19656ff5dbdeed8948eb7570089e12d5}\ .$$

Now the forgery is ready to be put together: it includes data $\mathtt{tbsCert}^*$, signature $\mathtt{C.sigValue}$, and algorithm specification "PKCS #1 SHA-256 With RSA Encryption" (where $h = \mathsf{SHA256}$ is substituted by $\tilde{h}$ on the user's device.) This is shown in Figure 23.