Highly-Effective Backdoors for Hash Functions and Beyond

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

April 5, 2024

Abstract

We study the possibility of schemes whose public parameters have been generated along with a backdoor. We consider the goal of the big-brother adversary to be two-fold: It desires utility (it can break the scheme) but also exclusivity (nobody else can). Starting with hash functions, we give new, strong definitions for these two goals, calling the combination high effectiveness. We then present a construction of a backdoored hash function that is highly effective, meaning provably meets our new definition. As an application, we investigate forgery of X.509 certificates that use this hash function. We then consider signatures, again giving a definition of high effectiveness, and showing that it can be achieved. But we also give some positive results, namely that for the Okamoto and Katz-Wang signature schemes, certain natural backdoor strategies are provably futile. Our backdoored constructions serve to warn that backdoors can be more powerful and damaging than previously conceived, and to help defenders and developers identify potential backdoors by illustrating how they might be built. Our positive results illustrate that some schemes do offer more backdoor resistance than others, which may make them preferable.

[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

"Backdoors" are a long-standing concern in cryptography. This paper asks, just how effective can backdoors be, in terms of the capabilities they give the big-brother adversary? We answer this via new, strong definitions, and ways to achieve them. Overall, our results serve as a warning that backdoors can be much more effective than previously conceived. As an introduction to the topic of backdoors, we start with Dual EC as a canonical example.

THE DUAL EC BACKDOOR. Dual EC is a pseudo-random number generator (PRNG) designed by the NSA and then standardized as NIST SP 800-90 and ANSI X9.82. It includes (as public parameters) a pair g, h of generators of an elliptic curve group. Shumow and Ferguson [45] pointed out a potential backdoor, specifically the discrete logarithm x of h to base g. They showed how to use x to predict PRNG outputs without having the seed, and this was later shown to allow attacks on TLS [13].

Could the NSA have generated g and $h = g^x$ so as to know and retain the backdoor x? The Snowden revelations (project Bullrun and SIGINT) suggest so. It also appeared that the NSA went to considerable efforts to ensure adoption and standardization of Dual EC, including (as per Reuters, 2013) paying RSA corporation $10 million to make Dual EC the default method for psuedo-random generation in their BSafe library. A full account of the Dual EC story is in [9].

THE SETTING. Research on backdoors, although present prior to Dual EC, intensified after it [2, 4, 6, 17, 19, 22]. It sits within a broader field of work on subversion [5, 7, 31, 36, 41, 46, 47, 52–54] that we will discuss in Section 2. Focusing for now on backdoors, let us begin with the model, clarifying in particular the premises, goals and assumptions.

The adversary in this domain is often called big-brother, or the big-brother adversary. It designs a scheme S that uses public parameters pk that have been generated with a backdoor bd, as $(pk, bd) \leftarrow\!\!\$\ \mathsf{S.BKg}$. Big-brother represents *power*, such as a government or Intelligence Agency. It is assumed to have the political clout and leverage to enable S, with pk, to be deployed (even standardized) and used, even in the face of potential suspicions, or evidence, of backdooring. But governments aren't the only concern. Big-brother could also be a large corporation targeting its employees.

BACKDOOR EFFECTIVENESS. As a framework to capture and understand both prior work and ours, one can see big-brother's goal, in designing a backdoored scheme S, as having two parts:

- Utility: Big-brother, through knowledge of the backdoor bd, should be able to violate security of S_{pk}, but
- Exclusivity: Others, meaning entities possessing pk but not bd, should *not* be able to violate security of S_{pk}.

Utility of course represents the main intent of big-brother in creating backdoors, but exclusivity is equally important. For example a big-brother government or Intelligence Agency wants to ensure that other governments and their Intelligence Agencies cannot violate security of S. A big-brother corporation wants to ensure that its competitors and the public cannot violate security of S.

The amalgam of utility and exclusivity is called *effectiveness*. It is a measure of the quality of the backdoored scheme from the point of view of big-brother.

BUILDING BACKDOORED SCHEMES. A foundational line of work gives particular, explicit constructions of effective backdoored schemes. In particular this has been done for PRGs [19] and collision-resistant (CR) hash functions [2, 22]. The intent and value of this work is to serve as a warning, highlighting what big-brother could potentially do and what risks or threats we face. As part of this, the works formalize notions of effectiveness that they show their schemes meet.

We revisit building backdoored schemes. We argue that the notions of effectiveness in prior work are limited. This, as we will see, is true for *both* utility *and* exclusivity. We will give new, stronger definitions of both. We then build backdoored schemes that (provably) achieve them. We call these schemes highly effective.

INTENT. It should go without saying that our intent, in providing highly-effective backdoored schemes, is *not* to help big brother. Rather, it is to help defenders and developers by, first, warning that the risks and threats from backdooring are even greater than indicated by prior work, and second, helping them identify backdoors by giving examples of how they may be built. Finally, our definitional frameworks may be of use for proving that certain schemes are resistant to (algorithmic) backdoor attempts.

BACKDOORING HASH FUNCTIONS. We start with collision-resistant hash functions and then move to other primitives. A backdoored hash function BH allows big-brother to generate a hashing key hk along with a backdoor bd as $(hk, bd) \leftarrow\!\!\$\ \mathsf{BH.BKg}$. The hashing key hk is public and defines a deterministic evaluation algorithm $\mathsf{BH.Ev}(hk, \cdot) : \{0,1\}^* \to \{0,1\}^{\mathsf{BH.ol}}$ that is the actual (public) hash function that everyone will use. A collision is a pair of distinct points e_1, e_2 such that $\mathsf{BH.Ev}(hk, e_1) = \mathsf{BH.Ev}(hk, e_2)$. The main security goal is collision resistance (CR). Effectiveness as per Fischlin, Janson and Mazaheri (FJM) [22] is quite natural, asking for:

- Basic Utility: Given hk, bd, one can efficiently violate CR, meaning produce some distinct e_1, e_2 such that $\mathsf{BH.Ev}(hk, e_1) = \mathsf{BH.Ev}(hk, e_2)$.
- Basic Exclusivity: An adversary given hk but not bd cannot violate CR.

We'll refer to the combination as basic effectiveness. FJM [22] build a hash function BH satisfying this. Roughly, hk is the image of bd under a one-way function F, and $\mathsf{BH.Ev}(hk, \cdot)$ behaves anomalously if (and only if) its input maps to hk under F. Albertini, Aumasson, Eichlseder, Mendel and Schläffer (AAEMS) [2] give a backdoored version of SHA1. Certain provably secure hash functions can also be viewed as backdoored hash functions achieving basic effectiveness. For example, the VSH (Very Smooth Hash) algorithm of Contini, Lenstra and Steinfeld [14] uses a public modulus $N = pq$; knowing the factorization allows one to find collisions (which then reveal $\varphi(N)$).

LIMITATIONS OF BASIC UTILITY. Basic utility allows big-brother to find *some* collision e_1, e_2. But in its quest for subversion, big-brother wants significantly more. We illustrate with two examples.

A commonly considered goal for big-brother in this setting [48, 49, 55] is forgery of TLS certificates allowing it to impersonate a website `legit.com`. Suppose the latter has a certificate CERT issued by a certificate authority CA. The certificate consists of data (content) CERT.D and the CA's RSA signature $\mathrm{SIG}_{\mathrm{CA}}$ on the hash $h = \mathsf{BH.Ev}(hk, \mathrm{CERT.D})$. The data includes `legit.com`'s public key D.PK, its identity D.ID = `legit.com`, and other auxiliary information. X.509 is the standard format for such certificates. Big-brother wants to construct a certificate $\overline{\mathrm{CERT}}$ that allows it to impersonate `legit.com`. Since it cannot forge the CA signature, it wants (using its backdoor for BH) to alter the data to $\overline{\mathrm{D}}$ such that $\mathsf{BH.Ev}(hk, \overline{\mathrm{D}}) = h$, allowing it to reuse the signature $\mathrm{SIG}_{\mathrm{CA}}$. But not just *any* choice of $\overline{\mathrm{D}}$ allowing this will do; there are significant constraints. First, the ID field $\overline{\mathrm{D}}$.ID must continue to be `legit.com`, so that the forged certificate allows impersonation of this website. Second, big-brother wants the public key $\overline{\mathrm{D}}$.PK to be one for which it knows the secret key, so that the forged certificate allows it to masquerade as `legit.com` in a TLS handshake.

The goal that emerges when given the backdoor is not to find some collision for $\mathsf{BH.Ev}(hk, \cdot)$ but rather, given a point h, to create a preimage e of h under $\mathsf{BH.Ev}(hk, \cdot)$ such that e embeds certain desired information in certain specific places in a way that allows the desired subversion.

Another example is password-based authentication. User U has password pwd. The server holds (h, s) where $h = \mathsf{BH.Ev}(hk, pwd \,\|\, s)$ and s is a public, random salt, as specified by PKCS#5 [33].

To authenticate itself, U sends pwd to the server (over a TLS connection), and the latter checks that $\mathsf{BH.Ev}(hk, pwd \,\|\, s) = h$. Assume (as is common) that the server's file of hashed passwords is compromised and big-brother knows h, s. To impersonate U, it wants to find $\overline{pwd}$ such that $\mathsf{BH.Ev}(hk, \overline{pwd} \,\|\, s) = h$. Again, the task (given the backdoor) is to find a preimage e under $\mathsf{BH.Ev}(hk, \cdot)$ of a target h such that e is not arbitrary but embeds some desired information (here the salt) in some desired way or place.

Beyond basic utility, FJM [22] and AAEMS [2] suggest that utility be the ability to violate preimage resistance. In the former this is formalized as: given $h = \mathsf{BH.Ev}(hk, x)$ for random x, the backdoor bd allows finding some x' satisfying $\mathsf{BH.Ev}(hk, x') = h$. But this will not suffice for the above subversion tasks because (1) big-brother wants to find, not *some* preimage x', but one that is constrained to have a particular structure, and (2) the x under which the given h was computed was not random but itself structured. We will accordingly formalize a new, high-utility definition that allows both the subversion tasks above.

Limitations of basic exclusivity. Having seen that basic utility is limited, we argue the same for basic exclusivity. Recall this asks that an adversary not having the backdoor bd be unable to violate CR. This fails to consider that collisions created by big-brother using bd can reveal the backdoor, making it easy to find further collisions. Indeed, this happens for basic constructions of backdoored schemes such as outlined above. We will address this by formalizing a high-exclusivity goal inspired by definitions of collision-resistance for chameleon hash functions given by Derler, Samelin and Slamanig (DSS) [18].

Our new definitions. We define high utility and high exclusivity, referring to the amalgam as high effectiveness. We first summarize the requirements and then discuss them. High utility below is parameterized by a predicate $\mathsf{P}(\cdot, \cdot)$ called the constraint:

- High Utility: There is an efficient algorithm $\mathsf{BH.FP}$ that given the backdoor bd, a string u called the constraint-parameter, and a target $h \in \{0,1\}^{\mathsf{BH.ol}}$, returns an e such that (1) $\mathsf{BH.Ev}(hk, e) = h$ and (2) $\mathsf{P}(e, u) = \mathsf{true}$.
- High Exclusivity: An adversary given hk and an oracle for $\mathsf{BH.FP}(bd, \cdot, \cdot)$ should not be able to find any non-trivial collision for $\mathsf{BH.Ev}(hk, \cdot)$.

In high utility, the constraint parameter u represents the information we want to embed in e, and the constraint predicate P checks that e embeds u correctly. In the case of certificate subversion, u would be the values of the fields $\overline{\mathrm{D}}$.ID and $\overline{\mathrm{D}}$.PK that big-brother wants the certificate data $\overline{\mathrm{D}}$ to contain, and the constraint predicate $\mathsf{P}_{\mathrm{cert}}$ would, given e, u, parse e as a certificate and check that the appropriate fields have values as given by u. For subversion of password-based authentication, u is the salt and $\mathsf{P}(e, u)$ checks that u is a suffix of e. In this way, through choices of P, we can capture a variety of subversion tasks.

High exclusivity now allows the adversary access to $\mathsf{BH.FP}(bd, \cdot, \cdot)$ as a way to capture possession of preimages created by big-brother in the past. The oracle of course trivially allows creation of some collisions, and the requirement is that the adversary not find further collisions. This in particular precludes an output of $\mathsf{BH.FP}$ revealing the backdoor. Saying precisely what it means for a collision to be "non-trivial" is delicate and our formal definition in the body of the paper is more fine-grained, giving the adversary an additional oracle.

The **HEB** construction. Is it possible to build highly-effective backdoored hash functions? We show, through construction, that the answer is "yes."

To expand on this, first note that one cannot hope to achieve high effectiveness for all P. It is, for example, impossible for the predicate $\mathsf{P}(e, u)$ that returns true iff $e = u$; intuitively, one needs some "room" in e for the backdoor to exploit. We show how to build a highly-effective backdoored

hash function BH 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 the $\mathsf{P}_{\mathrm{cert}}$ predicate allowing certificate subversion as well as predicates allowing subversion of password-based authentication.

Our construction is a transform **HEB** (Highly Effective Backdoor) that takes (1) a standard collision-resistant hash function G (2) a signature scheme S and (3) an *embedding function* Emb, for the target predicate P, that is compatible with S, G, as we will define in Section 4. (Embeddability, for now, just asks that such an embedding exists.) It returns a backdoored hash function $\mathsf{BH} = \mathbf{HEB}[\mathsf{G}, \mathsf{S}, \mathsf{Emb}]$ as in Figure 6. We exhibit an efficient preimage finding algorithm $\mathsf{BH.FP}$ for which we prove that high utility for P is achieved (Proposition 4.1). Theorem 4.2 proves high exclusivity of BH assuming collision-resistance of the starting hash function G and strong unforgeability of the signature scheme S.

Application. As an application of our theoretical results, we outline a PKI certificate forgery attack. We consider X.509-format certificates where the hash function is an instance of our above-discussed backdoored construction for the constraint predicate $\mathsf{P}_{\mathrm{cert}}$. Now, given a legitimate certificate CERT, we use the backdoor to create a forged certificate $\overline{\mathrm{CERT}}$ on big-brother's choice of certificate data. Section 6 describes this in detail. We note that this application relies crucially on high-utility as we have defined it.

Detectability. In the design of backdoored schemes, undetectability of backdooring has not been a goal, either in the models or in practice. Examination of the code or description of a scheme may (and usually will) allow one to deduce that a backdoor could exist. This is true for past work [2, 19, 22] and continues to be true for us. In the same vein, examination of a collision may reveal (as it does in our construction) that it was created using the backdoor.

The argument in favor of this perspective is that big-brother is an entity with power, capable of ensuring or mandating deployment, standardization and use of the scheme despite potential for, or even evidence of, backdooring. The story of Dual EC supports this perspective. That the design admitted a backdoor was noted early, but this did not prevent its widespread use. Nor did the further and clearer evidence of backdooring [12, 13] that emerged with time.

Relation to CHFs. Like backdoored hash functions (BHFs), Chameleon Hash Functions (CHFs) [35] admit a trapdoor that allows finding collisions. There is however a fundamental difference, namely that CHFs are *randomized*. This precludes their directly replacing conventional hash functions like $\mathsf{SHA256}$ in conventional usages like PKI or password hashing, a limitation to which BHFs, being deterministic, are not subject.

Nonetheless, at the technical and definitional levels, BHFs and CHFs are closely related. Accordingly we undertake in Section 5 a detailed study of the relations between the two primitives, using for this purpose the comprehensive definitions of CHFs of DSS [18]. We prove that a highly-effective BHF admits a correct (Proposition 5.4) and collision-resistant (Theorems 5.5, C.1) CHF, considering DSS's notions of either enhanced or full CR.

Backdoored signature schemes. The above outlook on backdooring can be brought to many primitives beyond hash functions. We illustrate with backdoored signatures. We define, for these, high utility and high exclusivity, as usual calling the combination high effectiveness. We then give a simple construction of a highly-effective backdoored signature scheme. In addition, and in a different vein, we also give positive results, showing that certain natural candidate backdoors fail for the Okamoto [39] and Katz-Wang [34] signature schemes. We now elaborate on these contributions.

A backdoored signature scheme BS allows generation of public parameters π along with a backdoor bd as $(\pi, bd) \leftarrow\!\!\$\ \mathsf{BS.BPg}$. Users generate their signing and verifying keys individually as

$(vk, sk) \leftarrow\!\!\$\ \mathsf{BS.Kg}(\pi)$. Basic utility might ask that possession of bd allows creation of some forgery, meaning a message m and a valid signature for it under the verification key vk of some user. We define high utility as asking for much more, namely that given any user verification key vk, and *any target message* m, the backdoor allows creation of a signature for m relative to vk. Likewise, basic exclusivity might simply ask that standard SUF-CMA security be maintained for adversaries not having bd, but we define high exclusivity as asking that this be true even with access to an oracle for producing backdoored signatures.

A natural way to define a backdoored signature scheme is to use our construction of a backdoored hash function. Due to the embeddability restriction on the constraint predicate, however, this will not allow backdoor-based forgery of arbitrary messages, failing high utility. We instead give a very simple alternative backdoored signature scheme which we prove achieves high effectiveness.

In the Okamoto signature scheme [39], the public parameters consist, like in Dual EC, of a pair g, h of group elements. The proof of security shows that forgery allows one to find the discrete logarithm x of h to base g. This leads to x being a natural candidate for a backdoor. Curiously, we show that knowledge of x does not allow backdooring. Our result (Theorem 8.1) is that Okamoto retains SUF-CMA security even against adversaries knowing x assuming SUF-CMA security of the Schnorr [44] signature scheme. We present a similar result for the Katz-Wang [34] signature scheme.

Postscript. This work was written prior to, and without knowledge of, the `xz` backdoor, discovered on March 29, 2024 [23]. While current understanding of the cryptographic portion of the backdoor [50] is different from our backdoored constructions, and from our X.509 application in Section 6, it also shows interesting similarities, 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 backdoors targeting high levels of efficacy are a realistic possibility, and motivates research, such as ours, on this topic. Once full details of `xz` are known, our framework may have further value, towards characterizing and understanding the capabilities and limitations of this backdoor, specifically by capturing it formally and assessing what definitions of utility, exclusivity, and potentially even indistinguishability, it meets.

2 Related work

Subverting cryptography. "Backdoors" are recurring concerns in cryptography. They have been observed in a variety of settings, including parameter generation in Dual EC [9,45], malicous code changes in Linux [21], and governmental exceptional access [1]. They have been imagined in many more. To situate our investigation of "backdoored hash functions," it is useful to consider three categories in this area of subverted cryptography. We summarize their respective notions of a "backdoor" in Figure 1.

In a first category, code can be maliciously modified from its algorithmic specification. This has been studied as algorithm substitution attacks (ASAs) [7] and as kleptography [52–54]. Here, an adversary's goal is to both modify an algorithm such that it exfiltrates secret information, and to keep this modification undetected. A limiting assumption in ASAs/kleptography is black-box-only access to algorithms. In a second category, backdoors are imagined as processes that allow an authority to access arbitrary secret keys. The authority can thus overcome usual security guarantees, but this extra power is well known to the public. Anamorphic cryptography [5,36,41] and earlier work on subliminal channels [31,46,47] have considered this type of extra-cryptographic subversion.

Third, and the category considered in this work, is maliciously designed algorithms or param-

	Goals	
	Subversion	Undetectability
ASAs/Kleptography	✓	✓
Exceptional access	✓	×
Basic algorithmic backdoors	✓	×
Highly-effective algorithmic backdoors	✓✓	×

Figure 1: Categories of subversion, and whether the subversion aims to be undetected. Note that a backdoor key could be kept secret while the *presence of a backdoor* is easily detected. The point of this comparison is to note that while such undetectability could improve a backdoor, it is not usually required in models or in practice.

---

eters. These have been studied for PRGs [17, 19], NIZKs [6], PKE [4], and hash functions (more on this below). Unlike an ASA, algorithms are assumed to be implemented honestly, and code can be inspected. Unlike anamorphic cryptography, public and secret keys are distributed as intended. However, this work considers backdoors that are stronger than prior work, along both dimensions of utility and exclusivity.

Hash functions. We now turn to existing work on hash functions. Definitions of backdoored hash functions have been considered by [2, 22] and we give a thorough comparison to the latter in Section 5. 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 [42] and from a proof-techniques perspective as programmable hash functions [30].

"Backdoored hash functions" also brings to mind asymmetric notions of hash functions. Chameleon hash functions were introduced by Krawczyk and Rabin in 2000 [35], and a recent overview and strengthening was given by [18]. Here, the backdoor key is used constructively for signature schemes and other applications; we include a detailed comparison to chameleon hash functions in Section 5. Another notion, that of trapdoor hash functions, was considered in 2019 by [20]. A trapdoor hash function is unkeyed, but includes a keyed hint function, with which certain bits of a hash preimage can be recovered. Trapdoor hash functions have found applications in secure two-party computation and despite the similar name, are different objects than backdoored hash functions.

A variety of prior work [48, 49, 55] has considered the implications of weak hash functions for PKI, a theme we also pursue.

3 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 message and output 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 $y \leftarrow A[\mathrm{O}_1, \ldots](x_1, \ldots; r)$.

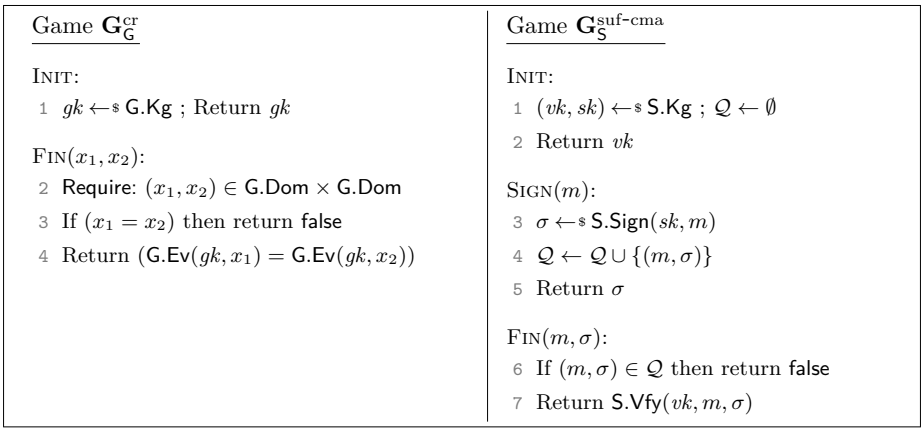

Figure 2: Collision resistance (left) and strong unforgeability (right).

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 $\langle \mathsf{Alg} \rangle$ to denote a description of algorithm Alg. 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 [8]. 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 Fin(output). 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.

Collision resistance. A hash function G consists of algorithms for key generation G.Kg and evaluation $\mathsf{G.Ev} : \mathrm{OUT}(\mathsf{G.Kg}) \times \mathsf{G.Dom} \to \{0,1\}^{\mathsf{G.ol}}$, where the domain of the hash function is G.Dom and the output length is G.ol. A standard security notion is the collision resistance of G, which is captured by game $\mathbf{G}^{\mathrm{cr}}_{\mathsf{G}}$ in Figure 2. If A is an adversary, we let $\mathbf{Adv}^{\mathrm{cr}}_{\mathsf{G}}(A) = \Pr[\,\mathbf{G}^{\mathrm{cr}}_{\mathsf{G}}(A)\,]$ be its cr advantage.

Unforgeability of signatures. A signature scheme S specifies algorithms S.Kg, S.Sign, S.Vfy, key spaces S.VK, S.SK, and signature length S.sl. Key generation 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 S.Vfy takes as input a

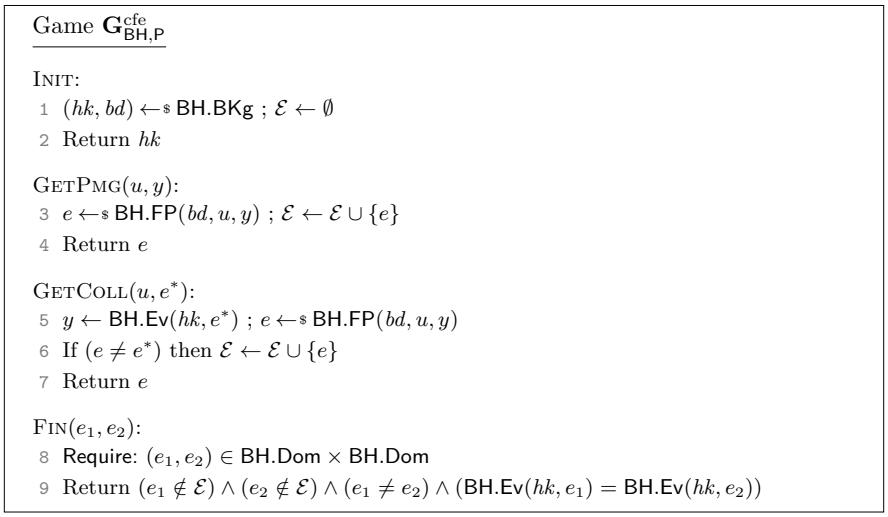

Figure 3: Collision-finding exclusivity (cfe) of a backdoored hash function BH.

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 $(vk, sk) \in \text{OUT}(\mathsf{S.Kg})$, 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 2. 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 [11], strong RSA [16, 26], and generally from one-way functions [28, Section 6.5].

4 Backdoored hash functions

Definitions. A backdoored hash function BH consists of four algorithms. Backdoored key generation $\mathsf{BH.BKg}$ returns a (public) hash key $hk \in \mathsf{BH.HK}$ and (private) backdoor $bd \in \mathsf{BH.BD}$ via $(hk, bd) \leftarrow\!\!\$\ \mathsf{BH.BKg}$. We define honest key generation $\mathsf{BH.Kg}$ to run $(hk, bd) \leftarrow\!\!\$\ \mathsf{BH.BKg}$ and return only hk. The deterministic hash evaluation function $\mathsf{BH.Ev} : \mathsf{BH.HK} \times \mathsf{BH.Dom} \to \{0,1\}^{\mathsf{BH.ol}}$ has domain $\mathsf{BH.Dom}$ a length-closed set and output hash length $\mathsf{BH.ol}$. Note that if we restrict attention to the pair of $\mathsf{BH.Kg}$ and $\mathsf{BH.Ev}$, these together form a standard hash function, for which one can consider cr security as in Section 3.

The *high utility* capabilities of BH are captured by its fourth algorithm, $\mathsf{BH.FP}$ ("find preimage") and by a constraint predicate P. Let $\mathsf{P} : \mathsf{BH.Dom} \times \{0,1\}^* \to \{\mathsf{true}, \mathsf{false}\}$ be a predicate. We say that BH achieves high utility relative to P, if for every constraint-parameter $u \in \{0,1\}^*$ and every $y \in \{0,1\}^{\mathsf{BH.ol}}$, if $e \leftarrow\!\!\$\ \mathsf{BH.FP}(bd, u, y)$ then we have (1) $\mathsf{BH.Ev}(hk, e) = y$ and (2) $\mathsf{P}(e, u) = 1$. In other words, the backdoor bd allows one to compute a preimage of any target hash, where the preimage also satisfies the constraint predicate. Such a notion strengthens basic backdoor notions, which we explain further below.

Recall that effectiveness of a backdoor asks for both utility and exclusivity. *High exclusivity* requires that a backdoored hash function remains collision-resistant to anyone without bd. This is captured by game $\mathbf{G}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}$ of Figure 3, where "cfe" denotes collision-finding exclusivity. If A is an adversary, we let $\mathbf{Adv}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}(A) = \Pr\left[\mathbf{G}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}(A)\right]$ be its cfe advantage. The difference from standard

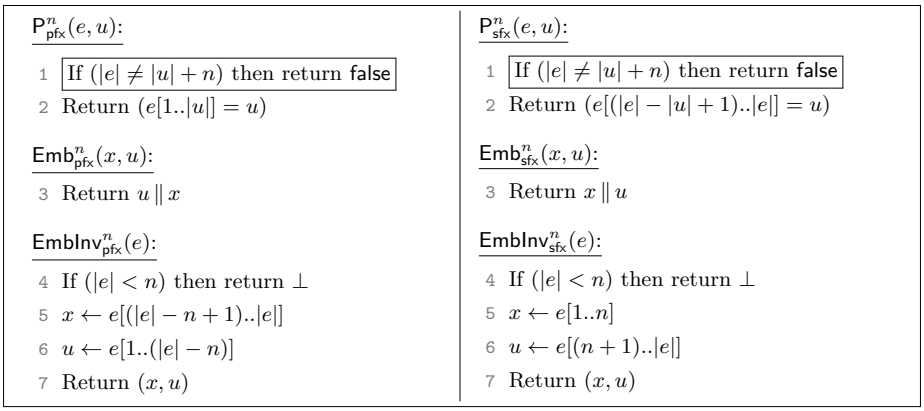

Figure 4: Practical examples of predicates and embedding functions. The embedding space is $\mathsf{Emb}^n_{\mathsf{pfx}}.\mathsf{ES} = \mathsf{Emb}^n_{\mathsf{sfx}}.\mathsf{ES} = \{0,1\}^n$ for a fixed n. The boxed code is not strictly required to have a correct embedding function for these predicates, but it will be required for a condition in Proposition 5.4 and more.

---

cr is the addition of the GetPmg, GetColl oracles, which allow an adversary to view preimages or second preimages, respectively, that have been produced by the backdoor. These are subject to adversary-chosen constraint-parameters u. An adversary A wins game $\mathbf{G}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}$ if it produces any collision which was not generated by BH.FP directly. The addition of these two oracles can be viewed as a formalization of "backdoor key exposure" as discussed in [22], which arises when *bd* is visible in outputs of BH.FP. We leave a full comparison to Section 5.

Predicates and embeddings. Let us now turn to some details of realizing constraints 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 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 $(x,u) \in \mathsf{Emb.ES} \times \{0,1\}^*$, if $e \leftarrow \mathsf{Emb}(x,u)$ then $\mathsf{P}(e,u) = 1$ and $\mathsf{Emb}^{-1}(e) = (x,u)$, and (2) $\mathsf{Emb}^{-1}(e) = \perp$ for all $e \notin \text{Im}(\mathsf{Emb})$. We say that Emb is a *correct embedding function* for predicate P if these two properties are satisfied.

Two illustrative examples are prefix and suffix embeddings. Suppose one wants to find a preimage e of hash y, with the constraint that e 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 constructing a backdoored hash function. In particular, setting $n = 0$ or $n = 1$ will not satisfy the conditions of Proposition 4.1 and Theorem 4.2, but larger values of n will.

A variety of predicates and embedding functions may be desirable in practice. Predicate $\mathsf{P}(e,u)$ could capture whether e is a valid X.509 certificate containing information u; this is considered in more detail in Section 6. A predicate could capture whether e can be parsed as human-readable text or otherwise does not "look suspicious." Increasingly useful predicates will come with implementation challenges beyond constructing a backdoored hash function BH, but the ones described above are already potent.

Warmup construction. We begin with a basic construction of a backdoored hash function. This is essentially the construction of [22, Section 7.1] in our syntax, without accounting for a predicate.

$\underline{\mathsf{BH}_0.\mathsf{Kg}}$:

1 $bd \leftarrow\!\!\$\ \{0,1\}^k$; $t \leftarrow \mathsf{F}(bd)$
2 $gk \leftarrow\!\!\$\ \mathsf{G.Kg}$; $hk \leftarrow (gk, t)$
3 Return hk

$\underline{\mathsf{BH}_0.\mathsf{Ev}(hk, e)}$:

4 $(gk, t) \leftarrow hk$
5 If $(|e| = k + \mathsf{G.ol})$ then
6 $\quad bd' \leftarrow e[1..k]$
7 $\quad$ If $(\mathsf{F}(bd') = t)$ then return $e[(k+1)..|e|]$
8 Else return $\mathsf{G}(gk, e)$

$\underline{\mathsf{BH}_0.\mathsf{BKg}}$:

9 $bd \leftarrow\!\!\$\ \{0,1\}^k$; $t \leftarrow \mathsf{F}(bd)$
10 $gk \leftarrow\!\!\$\ \mathsf{G.Kg}$; $hk \leftarrow (gk, t)$
11 Return (hk, bd)

$\underline{\mathsf{BH}_0.\mathsf{FP}(bd, u, y)}$:

12 // In the warmup, u is ignored
13 Require: $y \in \{0,1\}^{\mathsf{G.ol}}$
14 $e \leftarrow bd \,\|\, y$
15 Return e

Figure 5: Warmup construction of backdoored hash function BH_0.

$\underline{\mathsf{BH.Kg}}$:

1 $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
2 $gk \leftarrow\!\!\$\ \mathsf{G.Kg}$
3 $hk \leftarrow (gk, vk)$
4 Return hk

$\underline{\mathsf{BH.Ev}(hk, e)}$:

5 $(gk, vk) \leftarrow hk$
6 $w \leftarrow \mathsf{Emb}^{-1}(e)$
7 If $(w = \perp)$ then return $\mathsf{G}(gk, e)$
8 $((y \,\|\, \sigma), u) \leftarrow w$
9 If $\mathsf{S.Vfy}(vk, (y, u), \sigma)$ then return y
10 Else return $\mathsf{G}(gk, e)$

$\underline{\mathsf{BH.BKg}}$:

11 $(vk, sk) \leftarrow\!\!\$\ \mathsf{S.Kg}$
12 $gk \leftarrow\!\!\$\ \mathsf{G.Kg}$
13 $hk \leftarrow (gk, vk)$; $bd \leftarrow sk$
14 Return (hk, bd)

$\underline{\mathsf{BH.FP}(bd, u, y)}$:

15 Require: $y \in \{0,1\}^{\mathsf{BH.ol}}$
16 $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y, u))$
17 $e \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$
18 Return e

Figure 6: Construction of backdoored hash function BH for predicate P with associated embedding function Emb.

Given a one-way function $\mathsf{F} : \{0,1\}^k \to \{0,1\}^\ell$ and a cr hash function G, backdoored hash function BH_0 is specified by the component algorithms in Figure 5. BH_0 achieves some degree of utility and exclusivity. To an entity with bd, finding a preimage of target y is simple: $bd \,\|\, y$ is a preimage because the trigger $(\mathsf{F}(bd') = t)$ passes in $\mathsf{BH}_0.\mathsf{Ev}$, which then returns y. In terms of exclusivity, t is public but bd remains hard to find assuming F is one-way. We omit the analysis as we will next consider a highly effective construction. In particular, we would like to find preimages satisfying a predicate P, and achieve stronger cfe exclusivity.

Construction. Let us now turn to meeting these high effectiveness requirements of utility and cfe exclusivity. Let S be an suf-cma signature scheme and let G be a cr family of hash functions. We say that message embedding function $\mathsf{Emb} : \mathsf{Emb.ES} \times \{0,1\}^* \to \{0,1\}^*$ is *compatible* with S, G if $\mathsf{Emb.ES} = \{0,1\}^{\mathsf{G.ol}+\mathsf{S.sl}}$. That is, the embedding information consists of an output of G and a signature. Our highly-effective-backdoor transform **HEB** associates to S, G, and an Emb compatible to S, G a backdoored hash function $\mathsf{BH} = \mathbf{HEB}[\mathsf{G}, \mathsf{S}, \mathsf{Emb}]$ which is defined in Figure 6. We let $\mathsf{BH.HK} = \mathrm{OUT}(\mathsf{G.Kg}) \times \mathsf{S.VK}$ and $\mathsf{BH.BD} = \mathsf{S.SK}$, with hash evaluation function $\mathsf{BH.Ev} : \mathsf{BH.HK} \times \{0,1\}^* \to \{0,1\}^{\mathsf{BH.ol}}$. 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, G.

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

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

Proof of Proposition 4.1: Consider any $(hk, bd) \leftarrow\!\!\$\ \mathsf{BH.BKg}$, $u \in \{0,1\}^*$, and $y \in \{0,1\}^{\mathsf{BH.ol}}$. The function $\mathsf{BH.FP}$, on inputs bd,u, and y, returns $e \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$ where $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y, u))$. Property (1) of high utility requires that $\mathsf{BH.Ev}(hk, e) = y$. Let us consider $\mathsf{BH.Ev}(hk, e)$ of construction BH. On lines 6,7, since $e \in \mathrm{Im}(\mathsf{Emb})$, $w \neq \perp$. 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 9 runs $\mathsf{S.Vfy}(vk, (y, u), \sigma)$ where $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y, u))$. This passes as long as S is a correct signature scheme, and $\mathsf{BH.Ev}(hk, e)$ thus returns y on line 9.

Property (2) of high utility asks that $\mathsf{P}(e, u) = 1$. This is proven by line 17, where $e \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$. Correctness of the embedding function Emb for predicate P implies that $\mathsf{P}(e, u) = 1$. ∎

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

$$\mathbf{Adv}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}(A) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}}) + \mathbf{Adv}^{\mathrm{cr}}_{\mathsf{G}}(A_{\mathsf{G}})\ . \tag{1}$$

If A *makes* q_p GetPmg *queries and* q_c GetColl *queries then* A_{S} *makes* $(q_p + q_c)$ Sign *queries. The running times of* $A_{\mathsf{S}}, A_{\mathsf{G}}$ *are close to that of* A*.*

Proof of Theorem 4.2: Consider game G_0 of Figure 7. We claim that

$$\mathbf{Adv}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}(A) = \Pr[\,\mathrm{G}_0(A)\,]\ . \tag{2}$$

To justify Eq. (2), we claim that the $\textsc{Fin}(e_1, e_2)$ return value is the same in G_0 as it is in $\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}$. (The Init, GetPmg, and GetColl oracles are identical, instantiated with scheme BH.) In particular, the three checks made in Fin are identical. The first two checks are captured in lines 12,13 of G_0, which check whether $(e_1 = e_2)$ or if some $e_i \in \mathcal{E}$. For the third, we claim that on line 20 of G_0, it is the case that $y'_i = \mathsf{BH.Ev}(hk, e_i)$, meaning that G_0 checks whether $\mathsf{BH.Ev}(hk, e_1) = \mathsf{BH.Ev}(hk, e_2)$. Consider y'_i of G_0. Lines 15-19 of G_0 correspond to lines 6-10 of $\mathsf{BH.Ev}$ in Figure 6. That is, $y'_i = \mathsf{BH.Ev}(hk, e_i) = \mathsf{G}(gk, e_i)$ if there is no valid parsing of w nor verified signature, and $y'_i = \mathsf{BH.Ev}(hk, e_i) = y_i$ if the parsing and signature do pass. This proves Eq. (2).

We next turn to game G_1 and claim that

$$\Pr[\,\mathrm{G}_0(A)\,] \leq \Pr[\,\mathrm{G}_1(A)\,]\ . \tag{3}$$

The difference between games $\mathrm{G}_0, \mathrm{G}_1$ is the inclusion of line 13 versus 18 (the boxed code is present in both). To prove Eq. (3), we show that if $\mathrm{G}_0(A)$ outputs true, then $\mathrm{G}_1(A)$ outputs true. There

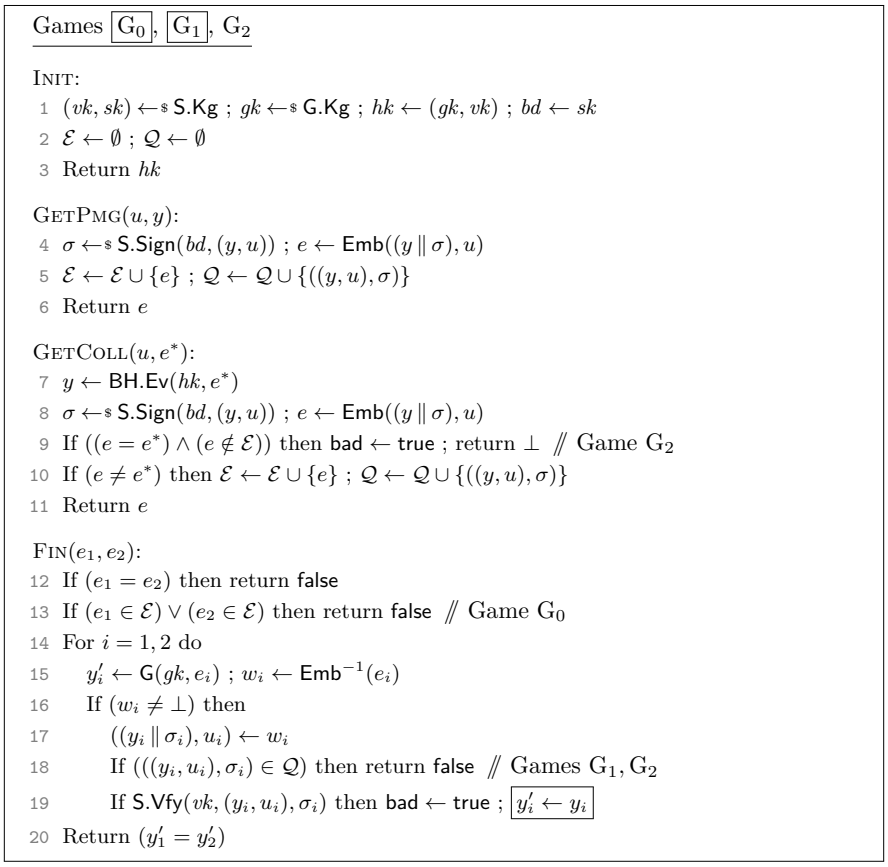

Figure 7: Games $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$ for the proof of Theorem 4.2. $\mathrm{G}_0, \mathrm{G}_1$ contain the boxed code and G_2 does not. Lines 13,18,9 are only present in $\mathrm{G}_0, \mathrm{G}_1, \mathrm{G}_2$, respectively.

---

are two cases. If $w_i = \perp$ for both i, then this follows immediately from the exclusion of line 13 in G_1. If $w_i \neq \perp$ for at least one i, then we claim that line 18 returns false only if line 13 would have also returned false. More precisely,

$$((y_i, u_i), \sigma_i) \in \mathcal{Q} \implies e_i \in \mathcal{E} \ .$$

Let us consider what is implied when $((y_i, u_i), \sigma_i) \in \mathcal{Q}$. The tuple must have been added to $\mathcal{Q}$ either on line 5 or 10. If the former, this must have occured on a $\textsc{GetPmg}(u_i, y_i)$ query where line 4 computed $\sigma_i \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y_i, u_i))$. The embedding then added to set $\mathcal{E}$ on line 5, call it e', was $e' \leftarrow \mathsf{Emb}((y_i \,\|\, \sigma_i), u_i)$. Thus $e' \in \mathcal{E}$. To show that in fact $e_i = e'$ we use the correctness of the embedding function. Concretely, we have both $((y_i \,\|\, \sigma_i), u_i) \leftarrow \mathsf{Emb}^{-1}(e_i)$ and $e' \leftarrow \mathsf{Emb}((y_i \,\|\, \sigma_i), u_i)$. This implies $e_i = e'$. For the second option, suppose the tuple was added to $\mathcal{Q}$ during a $\textsc{GetColl}(u_i, e^*)$ query. Then for some $e' \neq e^*$, line 8 computed $\sigma_i \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y_i, u_i))$ and $e' \leftarrow \mathsf{Emb}((y_i \,\|\, \sigma_i), u_i)$. Now $e' \in \mathcal{E}$ and by the argument above, we in fact have $e_i = e'$. This completes our justification that $((y_i, u_i), \sigma_i) \in \mathcal{Q} \implies e_i \in \mathcal{E}$, and thus of Eq. (3).

Let us now turn to G_2. Games $\mathrm{G}_1, \mathrm{G}_2$ are identical-until-bad, so by the Fundamental Lemma of Game Playing [8] we have

$$\begin{aligned} \Pr[\,\mathrm{G}_1(A)\,] &= \Pr[\,\mathrm{G}_2(A)\,] + (\Pr[\,\mathrm{G}_1(A)\,] - \Pr[\,\mathrm{G}_2(A)\,]) \\ &\leq \Pr[\,\mathrm{G}_2(A)\,] + \Pr[\,\mathrm{G}_2(A) \text{ sets } \mathsf{bad}\,] \ . \end{aligned}$$

Adversary $A_{\mathsf{S}}(vk)$:

1 $gk \leftarrow\!\!\$\ \mathsf{G.Kg}$; $hk \leftarrow (gk, vk)$; $\mathcal{E} \leftarrow \emptyset$
2 $(e_1, e_2) \leftarrow A[\textsc{GetPmg}_{\mathsf{S}}, \textsc{GetColl}_{\mathsf{S}}](hk)$
3 For $i = 1, 2$ do
4 $\quad w_i \leftarrow \mathsf{Emb}^{-1}(e_i)$
5 $\quad$ If $(w_i \neq \perp)$ then
6 $\quad\quad ((y_i \,\|\, \sigma_i), u_i) \leftarrow w_i$
7 $\quad\quad$ If $\mathsf{S.Vfy}(vk, (y_i, u_i), \sigma_i)$ then
8 $\quad\quad\quad$ Return $((y_i, u_i), \sigma_i)$

Oracle $\textsc{GetPmg}_{\mathsf{S}}(u, y)$:

9 $\sigma \leftarrow \textsc{Sign}((y, u))$
10 $e \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$
11 Return e

Oracle $\textsc{GetColl}_{\mathsf{S}}(u, e^*)$:

12 $w^* \leftarrow \mathsf{Emb}^{-1}(e^*)$
13 If $((w^* \neq \perp) \wedge (e^* \notin \mathcal{E}))$ then
14 $\quad ((y^* \,\|\, \sigma^*), u^*) \leftarrow w^*$
15 $\quad$ If $\mathsf{S.Vfy}(vk, (y^*, u^*), \sigma^*)$ then
16 $\quad\quad$ Return $\perp$
17 $\quad\quad$ Halt on output $((y^*, u^*), \sigma^*)$
18 $y \leftarrow \mathsf{BH.Ev}(hk, e^*)$
19 $\sigma \leftarrow \textsc{Sign}((y, u))$
20 $e \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$
21 If $(e \neq e^*)$ then $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$
22 Return e

Adversary $A_{\mathsf{G}}(gk)$:

1 $(vk, bd) \leftarrow\!\!\$\ \mathsf{S.Kg}$; $hk \leftarrow (gk, vk)$
2 $\mathcal{E} \leftarrow \emptyset$
3 $(e_1, e_2) \leftarrow A[\textsc{GetPmg}_{\mathsf{G}}, \textsc{GetColl}_{\mathsf{G}}](hk)$
4 Return (e_1, e_2)

Oracle $\textsc{GetPmg}_{\mathsf{G}}(u, y)$:

5 $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y, u))$
6 $e \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$; $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$
7 Return e

Oracle $\textsc{GetColl}_{\mathsf{G}}(u, e^*)$:

8 $y \leftarrow \mathsf{BH.Ev}(hk, e^*)$
9 $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y, u))$
10 $e \leftarrow \mathsf{Emb}((y \,\|\, \sigma), u)$
11 If $((e = e^*) \wedge (e \notin \mathcal{E}))$ then return $\perp$
12 If $(e \neq e^*)$ then $\mathcal{E} \leftarrow \mathcal{E} \cup \{e\}$
13 Return e

Figure 8: Adversaries A_{S} and A_{G} for the proof of Theorem 4.2.

We next construct adversaries $A_{\mathsf{S}}, A_{\mathsf{G}}$ for which we claim that

$$\Pr[\,\mathrm{G}_2(A) \text{ sets } \mathsf{bad}\,] \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}}) \tag{4}$$

$$\Pr[\,\mathrm{G}_2(A)\,] \leq \mathbf{Adv}^{\text{cr}}_{\mathsf{G}}(A_{\mathsf{G}})\ . \tag{5}$$

This will complete the proof of Eq. (1) and the theorem statement.

We begin by explaining adversary A_{S}, which is in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}$ and runs A as specified in Figure 8. We now make use of the two bad flags in game G_2 on lines 9,19. Suppose that $\mathsf{bad} \leftarrow \mathsf{true}$ on line 9 of G_2. Then there is a $\textsc{GetColl}(u, e^*)$ query such that $e^* \notin \mathcal{E}$ and $e^* = \mathsf{Emb}((y \,\|\, \sigma), u)$ where $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(bd, (y, u))$. Now let us consider lines 12-15 of oracle $\textsc{GetColl}_{\mathsf{S}}$. On this query, $w^* = \mathsf{Emb}^{-1}(e^*)$ and since e^* is an embedding, we can parse $((y^* \,\|\, \sigma^*), u^*) \leftarrow w^*$. Due to correctness of the embedding we have $y^* = y$, $\sigma^* = \sigma$, and $u^* = u$. Now verification on line 15 computes $\mathsf{S.Vfy}(vk, (y^*, u^*), \sigma^*) = \mathsf{S.Vfy}(vk, (y, u), \sigma)$, which passes if S is a correct signature scheme. In this scenario, $\mathrm{G}_2(A)$ sets bad and adversary A_{S} returns $\perp$, then halts on output $((y^*, u^*), \sigma^*)$. Since $e^* \notin \mathcal{E}$ we know that $((y^*, u^*), \sigma^*) \notin \mathcal{Q}$ (as A_{S} does not query $\textsc{Sign}$ until line 19). Thus $((y^*, u^*), \sigma^*)$ is a winning output in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}(A_{\mathsf{S}})$.

It could also be the case that $\mathsf{bad} \leftarrow \mathsf{true}$ on line 19 of G_2. Then A returned (e_1, e_2) such that $\mathsf{S.Vfy}(vk, (y_i, u_i), \sigma_i) = 1$ and $((y_i, u_i), \sigma_i) \notin \mathcal{Q}$. We may assume now that bad has not been previously set on line 9, else execution would have ended; this guarantees that all $\textsc{Sign}$ queries have

been appropriately added to the accounting set $\mathcal{Q}$. Therefore this $((y_i, u_i), \sigma_i)$ tuple is precisely a valid message-signature pair which wins game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}$. This completes the proof of Eq. (4). Note that A_{S} makes one SIGN query for each of A's GETPMG$_{\mathsf{S}}$ or GETCOLL queries, proving the running time in the theorem statement.

Finally, let us justify Eq. (5). Adversary A_{G}, which is in game $\mathbf{G}^{\text{cr}}_{\mathsf{G}}$, runs A as specified in Figure 8. A's view is again that of game G_2; initialization, GETPMG$_{\mathsf{G}}$, and GETCOLL$_{\mathsf{G}}$ return the same responses as in G_2. Now, if $\mathrm{G}_2(A)$ returns true, and since the boxed code is not executed in G_2, then it must be that $\mathsf{G}(gk, e_1) = \mathsf{G}(gk, e_2)$. This is precisely the winning condition of A_{G}'s game $\mathbf{G}^{\text{cr}}_{\mathsf{G}}$ and proves Eq. (5). A_{G} maintains running time close to that of A. ▌

5 Comparison to prior definitions

In the prior section we have seen that highly-effective backdoors are indeed possible to construct. This is not the first paper to consider backdoored or asymmetric hash functions; thus we can ask to what extent highly-effective backdoors are stronger than existing notions. We focus on two main notions. The first is basic backdoored hash functions, for which the most recent definitions have been given by FJM in 2018 [22]. AAEMS [2] gave similar, less formal notions in 2014, along with the concrete demonstration of an instantiation of $\mathsf{SHA1}$ with backdoored parameters. We visit this genre of basic backdoor definitions first, proving that the definition of a highly-effective backdoor is strictly stronger.

We then turn to chameleon hash functions (CHFs), which have seen significant research and applications. CHFs have a diverse array of security definitions, which have been summarized in [18]. There is a tradeoff in comparing a highly-effective backdoored hash function BH to a CHF CH: while $\mathsf{BH.Ev}$ is deterministic and operates like a "standard" hash function, certain CH constructions can achieve notions of indistinguishability. To clarify the difference, we show that one can construct a CHF CH from BH, such that CH satisfies correctness and collision resistance, but not indistinguishability. However, stepping back to the motivation for BH, we have not required a notion of indistinguishability. Instead, we require that $\mathsf{BH.Ev}$ is deterministic and maintains hash function syntax; this is not true of CHFs. Thus there are similarities (as we prove) but the applications are inherently different, and result in different achieved goals.

RELATION TO FJM [22]. We begin with basic backdoored hash functions as defined by Fischlin, Janson, and Mazaheri at CSF 2018 [22]. We provide their notions in our syntax here, and highlight differences. Call an FJM backdoored hash function FH. This includes a key-generation algorithm $\mathsf{FH.Kg}$ which generates a (public) hash key $\mathit{fhk} \in \mathsf{FH.HK}$ via $\mathit{fhk} \leftarrow\!\!\$\ \mathsf{FH.Kg}$, and a deterministic hash function $\mathsf{FH.Ev} : \mathsf{FH.HK} \times \mathsf{FH.Dom} \to \{0,1\}^{\mathsf{FH.ol}}$. Backdoored key-generation algorithm $\mathsf{FH.BKg}$ produces $(\mathit{fhk}, \mathit{fbd}) \leftarrow\!\!\$\ \mathsf{FH.BKg}$. However, there is no specified preimage-finding $\mathsf{FH.FP}$ algorithm.

Remark 5.1 *Technically,* $\mathsf{FH.BKg}$ *in FJM returns algorithm descriptions, not keys. That is,* $\mathsf{FH.BKg}$ *produces* $(\langle \mathsf{FH.Kg} \rangle, \langle \mathsf{FH.Ev} \rangle, \mathit{fbd}, r) \leftarrow\!\!\$\ \mathsf{FH.BKg}$. *In their stronger notion,* $r = \varepsilon$*, which is the case we consider. To simplify notation, we equivalently consider* $\langle \mathsf{FH.Kg} \rangle$ *and* $\langle \mathsf{FH.Ev} \rangle$ *to be captured by public key fhk.*

FJM consider collision resistance, preimage resistance, and second preimage resistance as properties that a backdoor could undermine. Utility and exclusivity are defined in terms of games which are given in our syntax in Figure 9. For each of $\mathrm{x} \in \{\text{f-cr}, \text{f-pr}, \text{f-2pr}\}$, backdoor utility is captured by game $\mathbf{G}^{\mathrm{x}}_{\mathsf{FH},1}$. The associated advantage of an adversary A is $\mathbf{Adv}^{\mathrm{x}}_{\mathsf{FH},1}(A) = \Pr\left[\mathbf{G}^{\mathrm{x}}_{\mathsf{FH},1}(A)\right]$. FJM

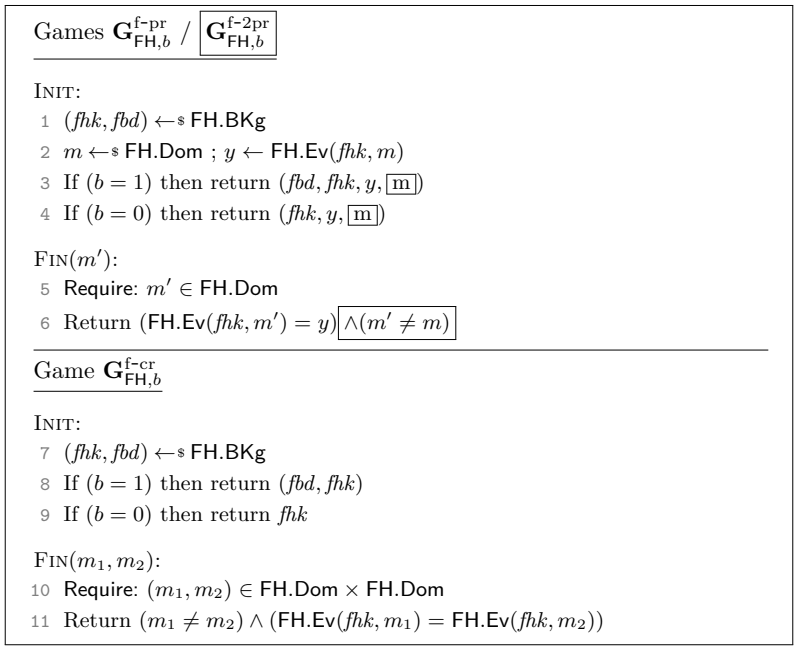

Figure 9: FJM exclusivity ($b = 0$) and backdoor utility ($b = 1$) for f-pr, f-2pr, and f-cr. The boxed code indicates the difference between first and second preimage resistance.

---

say that a backdoor achieves *basic utility* if there exists a p.p.t. adversary B such that $\mathbf{Adv}^{\text{x}}_{\mathsf{FH},1}(B)$ is nonnegligible. For each of x $\in$ {f-cr, f-pr, f-2pr}, *basic exclusivity* is then captured by game $\mathbf{G}^{\text{x}}_{\mathsf{FH},0}$. The associated advantage of an adversary A is $\mathbf{Adv}^{\text{x}}_{\mathsf{FH},0}(A) = \Pr\left[\,\mathbf{G}^{\text{x}}_{\mathsf{FH},0}(A)\,\right]$. Basic exclusivity (for f-cr, f-pr, or f-2pr) asks that $\mathbf{Adv}^{\text{x}}_{\mathsf{FH},0}(A)$ is negligible for all p.p.t. adversaries A.

A highly-effective backdoored hash function BH as presented in Section 4 is stronger than a basic backdoored hash function FH along a few dimensions. A quick sketch of differences is as follows. First, in terms of utility, BH includes an algorithm $\mathsf{BH.FP}$ which finds preimages of any target y with probability 1; this is in contrast to nonnegligible probability as in games $\mathbf{G}^{\text{x}}_{\mathsf{FH},1}$ of Figure 9. Moreover, if BH achieves utility with respect to P, then preimages produced by $\mathsf{BH.FP}$ are guaranteed to satisfy an additional predicate, a constraint which is not captured in basic utility. In terms of exclusivity, cfe includes GetPmg and GetColl oracles, which grant an adversary additional powers to view outputs of $\mathsf{BH.FP}$. Constructions like $\mathsf{BH_0}$ of Section 4 satisfy basic exclusivity but the backdoor is revealed by one query to GetPmg. Finally, FJM only consider one of x $\in$ {f-cr, f-pr, f-2pr} at a time. For example, FH is said to be effective if it achieves f-cr utility and f-cr exclusivity, but different goals are not simultaneously considered. (Why not f-2pr utility and f-cr exclusivity?) The notion of Section 4 asks for cfe exclusivity since that is strongest, and asks for preimage-finding utility, since that is more powerful than finding a collision.

Formally, given BH which is highly effective, we can construct FH which satisfies basic utility and exclusivity as defined by FJM. We consider the most powerful notions of utility (f-2pr, f-pr) and the strongest notion of exclusivity (f-cr). These are stated precisely and proved below. The other direction does not hold, with a justification that follows. Thus high effectiveness is indeed capturing a new, stronger aspect of backdoors.

$\underline{\mathsf{BH} \implies \text{FJM.}}$ We first prove that given a highly effective BH, we can construct an FJM backdoored hash function FH. This requires two statements. First, if BH achieves high utility then FH

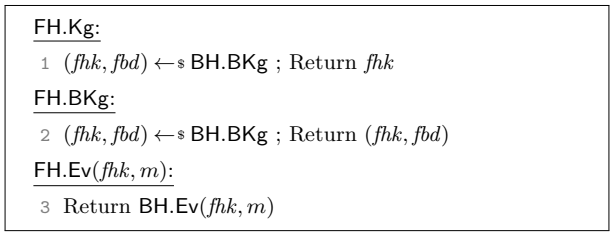

Figure 10: Construction of FJM backdoored hash function FH from highly-effective BH.

achieves utility (Proposition 5.2). Second, if BH meets cfe exclusivity then FH meets f-cr exclusivity (Theorem 5.3). We now proceed to the formal statements.

Proposition 5.2 *Let* BH *be a backdoored hash function. From* BH *we construct* FH *as in Figure 10. If* BH *achieves high utility for any predicate* P *then* FH *achieves f-pr utility, meaning that we give a p.p.t. adversary* B_1 *with* $\mathbf{Adv}^{\text{f-pr}}_{\mathsf{FH},1}(B_1) = 1$*. Moreover, if* P *is such that* $(\mathsf{P}(e,u) = \mathsf{true}) \implies (e \neq u)$*, then* FH *achieves f-2pr utility, meaning that we give a p.p.t. adversary* B_2 *achieving* $\mathbf{Adv}^{\text{f-2pr}}_{\mathsf{FH},1}(B_2) = 1$*. Two examples of such a* P *are the prefix and suffix predicates, with* $n > 0$*, of Figure 4.*

The above proposition says that the constructed FH achieves f-pr utility regardless of predicate, and achieves f-2pr utility for most predicates. In particular, the only requirement is that $\mathsf{P}(e,e) = \mathsf{false}$, which constrains BH.FP to find a *second* preimage. The proof of Proposition 5.2 is straightforward and is provided in Appendix A. We next provide the formal statement of exclusivity.

Theorem 5.3 *Let* BH *be a backdoored hash function and* P *any predicate. From* BH *we construct* FH *as in Figure 10. If* B *is an adversary in game* $\mathbf{G}^{\text{f-cr}}_{\mathsf{FH},0}$*, then we can construct adversary* A *in game* $\mathbf{G}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}$ *such that*

$$\mathbf{Adv}^{\text{f-cr}}_{\mathsf{FH},0}(B) \leq \mathbf{Adv}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}(A)\ . \tag{6}$$

Adversary A *has running time close to that of* B *and makes zero* GetPmg, GetColl *queries.*

The proof of Theorem 5.3 is similarly straightforward and in Appendix A. With this theorem, we complete the justification that a highly-effective BH implies an effective FH as defined by FJM, showing that the former is at least as strong as the latter. We next turn to showing that it is in fact strictly stronger.

FJM $\not\Rightarrow$ BH. The other direction does not hold, which we now explain. The first reason in syntactic; BH requires an algorithm BH.FP to be specified but FH of FJM does not. One might work around this by noting that if FH achieves basic utility (with respect to preimages) then there is an adversary B which achieves nonnegligible advantage $\mathbf{Adv}^{\text{f-pr}}_{\mathsf{FH},1}(B)$. Then one could define:

BH.FP(bd, u, y):
$m' \leftarrow B(fbd, fhk, y)$; Return m'

With some probability it is the case that $\mathsf{FH.Ev}(fhk, m') = y$, meaning that $\mathsf{BH.Ev}(hk, m') = y$ if we define BH.Ev to simply evaluate FH.Ev. However, FJM does not require this probability to be 1 for all y, and there is no syntax for requiring a predicate $\mathsf{P}(m', u)$ to be satisfied. Thus high utility of BH is not achieved, even for the vacuous predicate $\mathsf{P}(m', u)$ which always returns true.

Moreover, the exclusivity goal is different. While FJM ask that FH be collision-resistant to anyone without *fbd*, cfe exclusivity asks that BH be collision-resistant with the addition of GetPmg and GetColl oracles. The warmup construction $\mathsf{BH_0}$ in Figure 5 provides an explicit separation. This was proved to satisfy basic exclusivity (in the sense of f-cr, f-pr, or f-2pr) in [22, Section 7.1] as long as the underlying hash function G is correspondingly secure. However, let P always return true and recall that $\mathsf{BH_0.FP}(bd, u, y)$ returns $e \leftarrow bd \,\|\, y$. An adversary A with one query in game $\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH_0},\mathsf{P}}$ can choose y and obtain $e \leftarrow \textsc{GetPmg}(\varepsilon, y)$. An e produced using bd is of the form $bd \,\|\, y$ and thus bd is learned by A. Now A can produce a fresh collision using bd and achieve $\mathbf{Adv}^{\mathrm{cfe}}_{\mathsf{BH_0},\mathsf{P}}(A) = 1$.

This completes the analysis of highly-effective backdoors in comparison to basic effective backdoors: the new notion of high effectiveness is strictly stronger than existing definitions. Next we turn to a differently motivated, and well researched, category of asymmetric hash function.

Relation to Chameleon Hash Functions. CHFs may be motivated more by constructive applications than by fears of subversion, but they share similarities with backdoored hash functions. Introduced by Krawczyk and Rabin in 2000 [35], we pull from the recent definitions of Derler, Samelin, and Slamanig at PKC 2020 [18]. A chameleon hash function CH consists of four algorithms (and public parameters). Key generation produces a public and secret key via $(pk, sk) \leftarrow\!\!\$\ \mathsf{CH.Kg}$. Randomized hash evaluation takes as input a message m to produce $(h, r) \leftarrow\!\!\$\ \mathsf{CH.Hash}(pk, m)$, where h is considered the hash digest and r is a check value. Validation of a hash digest h and check value r is computed via the deterministic algorithm $d \leftarrow \mathsf{CH.Check}(pk, m, r, h)$, where $d \in \{0, 1\}$ indicates whether validation was successful. The "chameleon" notion is captured by algorithm $\mathsf{CH.Adapt}$ which computes a new check value via $r' \leftarrow\!\!\$\ \mathsf{CH.Adapt}(sk, m, m', r, h)$. Correctness of CH asks that for all $(pk, sk) \in \mathrm{OUT}(\mathsf{CH.Kg})$, for all $m, m' \in \mathsf{CH.Dom}$, for all $(h, r) \in \mathrm{OUT}(\mathsf{CH.Hash}(pk, m))$, for all $r' \in \mathrm{OUT}(\mathsf{CH.Adapt}(sk, m, m', r, h))$, it is the case that $\mathsf{CH.Check}(pk, m, r, h) = \mathsf{CH.Check}(pk, m', r', h) = 1$.

Security of a chameleon hash function is captured by a variety of notions, which depend on the application. In all cases, collision resistance is expected, but the definition varies. In many cases, indistinguishability is also expected, meaning that it is hard to tell whether (h, r) is an output of $\mathsf{CH.Hash}$ or has been computed with the aid of $\mathsf{CH.Adapt}$. The notion of a backdoored hash function BH in this paper is different than that of chameleon hash functions; it is not strictly stronger or weaker. Below, we show that given BH one can construct a chameleon hash function CH satisfying correctness and collision resistance, although not indistinguishability. In combination with Proposition 4.1 and Theorem 4.2, this gives rise to a chameleon hash function from OWFs and CRHFs.

However, let us step back to the motivation of BH. Given that CHFs exist and have been instantiated in myriad constructions, why wouldn't a backdooring entity simply mandate use of a CHF? We contend that the goals of big-brother are better achieved by simpler, deterministic constructions, which use conventional hash function syntax. What about indistinguishability? While this is not ruled out by a backdoored hash function BH, the question becomes whether a message m looks contrived. However, a larger question is the following. If a scheme is obviously backdoored, what additional power does distinguishing individual outputs provide to users? In some cases, users could reject a hash that they determine to be produced by the backdoor. In other cases, in particular if a scheme is standardized, there may not be a choice regardless. We contend that both cases are worth study, and the focus of this paper is on the latter.

Nonetheless, the goals of highly effective backdoors and CHFs overlap. We now present this formally, proving that given highly-effective BH we can construct a CHF CH that achieves correctness and collision resistance.

$\mathsf{BH} \implies \mathsf{CH}$ (with collision resistance). Given BH we now show how to construct CH which

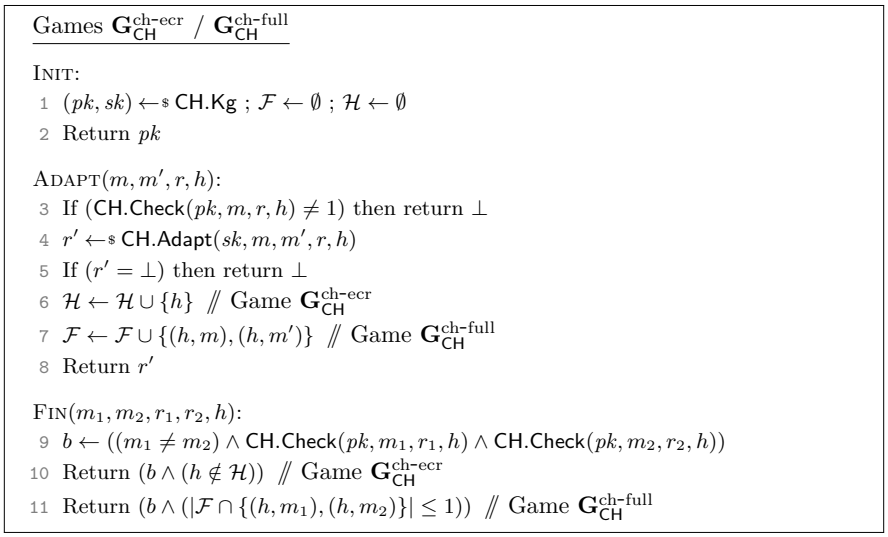

Figure 11: Full and enhanced collision resistance of a chameleon hash function CH, as defined by [18]. Lines 6,10 are only included in game $\mathbf{G}^{\text{ch-ecr}}_{\mathsf{CH}}$ while lines 7,11 are only included in game $\mathbf{G}^{\text{ch-full}}_{\mathsf{CH}}$.

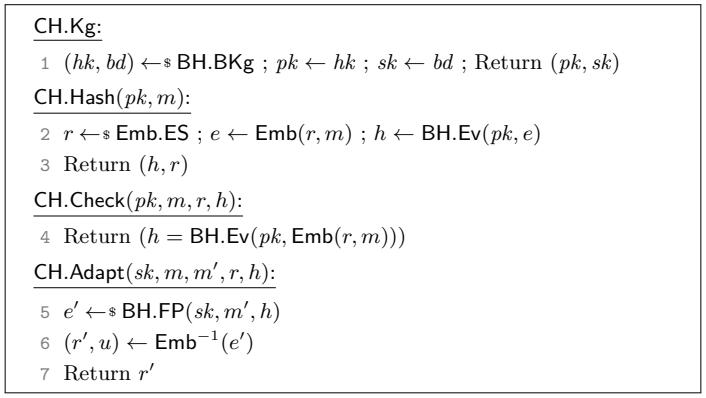

Figure 12: Construction of CH from BH for the proofs of Proposition 5.4, Theorem 5.5, and Theorem C.1 (full cr).

satisfies correctness and collision resistance. In particular, we can achieve "enhanced" collision resistance immediately, and the strongest notion of "full" collision resistance with an additional min-entropy assumption. We include the definitions of [18] as games $\mathbf{G}^{\text{ch-ecr}}_{\mathsf{CH}}$,$\mathbf{G}^{\text{ch-full}}_{\mathsf{CH}}$ of Figure 11. If A is an adversary, we let $\mathbf{Adv}^{\text{ch-ecr}}_{\mathsf{CH}}(A) = \Pr\left[\mathbf{G}^{\text{ch-ecr}}_{\mathsf{CH}}(A)\right]$ be its ch-ecr (enhanced cr) advantage and we let $\mathbf{Adv}^{\text{ch-full}}_{\mathsf{CH}}(A) = \Pr\left[\mathbf{G}^{\text{ch-full}}_{\mathsf{CH}}(A)\right]$ be its ch-full (full cr) advantage. We state the enhanced cr result below, for which the proof is simple, and provide the proof in Appendix B. The full cr result requires more setup to formally state, and we do so along with a proof in Appendix C.

Proposition 5.4 *Let* BH *be a backdoored hash function. From* BH *we construct chameleon hash function* CH *as in Figure 12. Let* P *be a predicate with a correct embedding function* Emb *such that* $(\mathsf{P}(e, u) = \mathsf{true}) \implies (\mathsf{Emb}^{-1}(e) = (x, u))$ *for some* $x \in \mathsf{Emb.ES}$*. For example, this is satisfied by the prefix and suffix embeddings in Figure 4. If* BH *achieves high utility for* P *then* CH *is correct.*

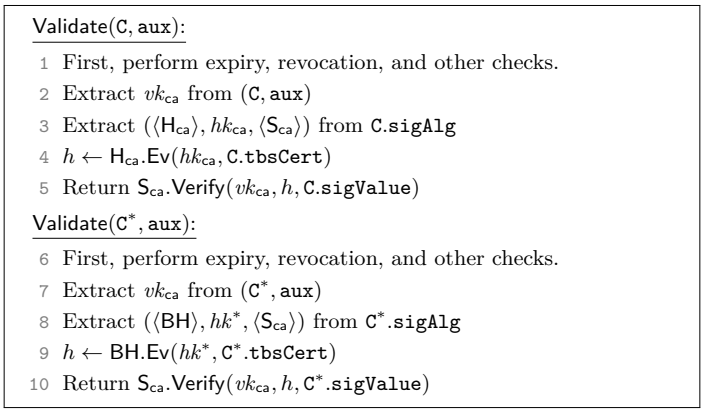

Figure 13: Validation of an X.509 certificate, simplified. Validate may be run on an honest certificate C (above) or a backdoored certificate C* (below).

Theorem 5.5 *Let* BH *be a backdoored hash function. From* BH *we construct chameleon hash function* CH *as in Figure 12. Let* P *be a predicate with a correct embedding function* Emb *such that* $(\mathsf{P}(e,u) = \mathsf{true}) \implies (\mathsf{Emb}^{-1}(e) = (x,u))$ *for some* $x \in \mathsf{Emb.ES}$*. Suppose* BH *achieves high utility for* P*. Then if* B *is an adversary in game* $\mathbf{G}^{\text{ch-ecr}}_{\mathsf{CH}}$*, we can construct adversary* A *in game* $\mathbf{G}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}$ *such that*

$$\mathbf{Adv}^{\text{ch-ecr}}_{\mathsf{CH}}(B) \leq \mathbf{Adv}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}(A)\ . \tag{7}$$

If B *makes* q ADAPT *queries then* A *makes* q GETPMG *queries. Adversary* A *has running time close to that of* B*.*

6 Application: Forged certificates

CERTIFICATES AND PKI. Hash functions are prevalent building blocks in a variety of schemes, one of which is public-key infrastructure (PKI) as realized by X.509 certificates and certificate authorities (CAs). For simplicity, suppose there is one honest CA who is operating with signature scheme S_{ca} and hash function SHA256. Let $(vk_{\mathsf{ca}}, sk_{\mathsf{ca}})$ be the verification and signing keys of the CA. As specified in RFC 5280 [15], a certificate C consists of a sequence of key-value pairs, some of which are mandatory. 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."
- 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 $\mathtt{tbsCert}'$ 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 application, we are more interested in validation than issuance, as we explain below. We consider the C.sigAlg field to consist of scheme descriptions $\langle \mathsf{H}_{\mathsf{ca}} \rangle$, $\langle \mathsf{S}_{\mathsf{ca}} \rangle$ and public parameter hk_{ca}.

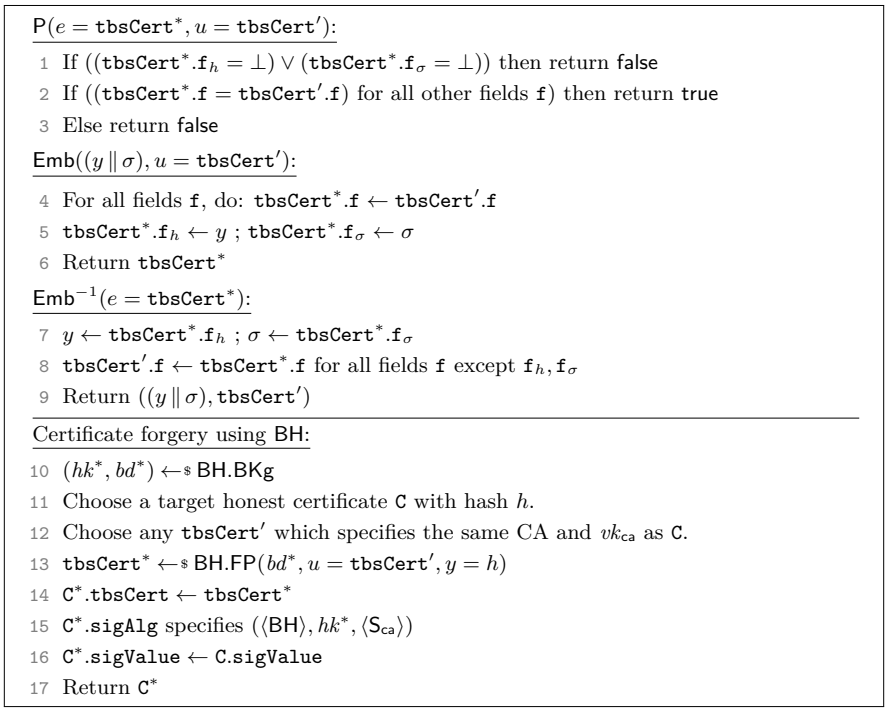

Figure 14: Predicate P and embedding Emb used to forge X.509 certificates (above). How a backdooring entity uses BH to forge certificates (below). We use $\mathtt{f}$ to denote any X.509 field name. Field labels $\mathtt{f}_h, \mathtt{f}_\sigma$ are fixed by the big-brother adversary to embed a target hash and big-brother signature, respectively, into a certificate, and are known in the algorithms of BH.

For common hash functions, hk_{ca} may be empty, but $\langle \mathsf{H}_{\mathsf{ca}} \rangle$ and hk_{ca} can together be considered to encapsulate an algorithm like SHA256. At a high level, validation Validate proceeds as in Figure 13.

Backdoor threat model. We assume that an honest CA is operating with signature scheme S_{ca} and hash function $\mathsf{H}_{\mathsf{ca}} = \mathsf{SHA256}$, neither of which is backdoored or otherwise known to be weak. We assume there is a public certificate $\mathtt{C}$ which has been signed by the CA. Let $h = \mathsf{H}_{\mathsf{ca}}.\mathsf{Ev}(hk_{\mathsf{ca}}, \mathtt{C.tbsCert})$; h is publicly computable. These are all standard aspects of PKI. The threat model for a backdoored hash function newly supposes that a hash function BH (with a particular public parameter hk^*) is an accepted algorithm by Validate for at least some client. The goal of the backdooring entity is for arbitrary certificate data $\mathtt{tbsCert}'$, to find certificate $\mathtt{C}^*$ such that $\mathsf{Validate}(\mathtt{C}^*, \mathtt{aux}) = 1$ for honest $\mathtt{aux}$, where $\mathtt{C}^*.\mathtt{tbsCert}$ is "close to" the intended $\mathtt{tbsCert}'$.

In real-world systems, weak hash functions have been exploited to find carefully constructed collisions [37, 38, 49, 51, 55]. At Eurocrypt 2007 [48], 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" [48]. The backdoor threat model supposes a big-brother adversary who wants to do even better: to forge (almost) arbitrary certifcates, at arbitrary times, and to do so easily.

Instantiation of a backdoored hash function. To achieve this end, a big-brother adversary uses a highly effective backdoored hash function, in particular one for the predicate P in Figure 14. This can be instantiated via the transform $\mathsf{BH} = \mathbf{HEB}[\mathsf{G}, \mathsf{S}, \mathsf{Emb}]$. As G we use SHA256, or whichever function is in use as H_{ca}. As S we use an suf-cma signature scheme with *short* signatures,

such as BLS [10]. We use the embedding function Emb given in Figure 14. At a high level, the specification in Figure 14 then allows a big-brother adversary to forge the following, given an honest certificate $\mathtt{C}$: for any desired $\mathtt{tbsCert}'$, they use $\mathsf{BH.FP}$ with bd^* to find $\mathtt{tbsCert}^*$ which is identical to $\mathtt{tbsCert}'$ except for two additional fields. They then build certificate $\mathtt{C}^*$ which includes data $\mathtt{tbsCert}^*$, specifies BH with hk^* as the hash algorithm, and copies the signature on $\mathtt{C}$. If BH is highly effective for predicate P, and if $\mathsf{Validate}(\mathtt{C}, \mathtt{aux}) = 1$, then $\mathsf{Validate}(\mathtt{C}^*, \mathtt{aux}) = 1$. Figure 13 shows the sequence of $\mathsf{Validate}$ steps on forgery $\mathtt{C}^*$.

The above shows how a certificate forgery can work, but a few remarks are in order. First, is it noticeable? Yes; $hk^* = (gk^*, vk^*)$ is public and anyone looking at $\mathtt{C}^*$ can parse $\mathsf{Emb}^{-1}(\mathtt{C}^*.\mathtt{tbsCert})$ to see $y, \sigma, \mathtt{tbsCert}'$ where $\mathsf{S.Vfy}(vk^*, (y, \mathtt{tbsCert}'), \sigma) = 1$. However, care could be taken to make the forgery less noticeable. Specifically, there may be more subtle ways to embed $(\mathsf{G.ol} + \mathsf{S.sl})$ extra bits into a certificate, but we have presented a simple case of inserting them into named fields $\mathtt{f}_h, \mathtt{f}_\sigma$. These could be $\mathsf{X.509}$ extension fields, for example, so may be overlooked. Second, what about the expiry and revocation checks in $\mathsf{Validate}$? We claim that if $\mathtt{tbsCert}^*$ is arbitrary save for two fields $\mathtt{f}_h, \mathtt{f}_\sigma$, then the validity dates and other information can be easily chosen to pass these checks. Third and finally, what does such a forged certificate actually look like? We present a more thorough example in Appendix D.

Remark 6.1 *It has been suggested that for backdoored symmetric primitives such as hash function components [22] and PRGs [17, 19], usage of asymmetric schemes will be noticed or impractical due to an obvious timing difference. The* $\mathsf{X.509}$ *validation process is interesting in that a signature verificaton is computed anyway; an additional verification inside the hash function becomes less drastic. However, real-life events [23] suggest that timing detection is still possible.*

7 Backdoored signatures

Inspired by this application, we now turn to signatures. A backdoored signature scheme should allow the big-brother adversary to forge signatures. For basic utility, one might start with asking for the ability to create *some* valid signature. Indeed, the application presented in the previous section already implicitly contains a backdoored signature scheme with basic utility (via a hash-then-sign design). It allows one with bd to forge signatures on messages that are restricted by the predicate embedding, assuming access to one honest signature.

In this section, we give definitions for backdoored signatures with high effectiveness. Intuitively, high utility should allow the owner of the backdoor to forge signatures for arbitrary messages, while high exclusivity asks that the scheme remains unforgeable even if one can observe signatures created by the backdoor. We start with these definitions and then turn to a construction meeting those.

DEFINITIONS. A backdoored signature scheme BS consists of six algorithms. Backdoored parameter generation $\mathsf{BS.BPg}$ returns public parameter π and (private) backdoor bd via $(\pi, bd) \leftarrow\!\!\$\ \mathsf{BS.BPg}$. Similar to our definition of backdoored hash functions, we define honest parameter generation $\mathsf{BS.Pg}$ to run $(\pi, bd) \leftarrow\!\!\$\ \mathsf{BS.BPg}$ and return only π. Key generation $\mathsf{BS.Kg}$ returns a verification and signing key via $(vk, sk) \leftarrow\!\!\$\ \mathsf{BS.Kg}(\pi)$. Signing takes as input a signing key sk and message $m \in \{0,1\}^*$ to return a signature via $\sigma \leftarrow\!\!\$\ \mathsf{BS.Sign}(\pi, sk, vk, m)$. Deterministic algorithm $\mathsf{BS.Verify}$ takes as input a verification key vk, message $m \in \{0,1\}^*$, and signature σ to return a bit d via $d \leftarrow \mathsf{S.Vfy}(\pi, vk, m, \sigma)$. Additionally, a backdoored signature scheme specifies an algorithm $\mathsf{BS.BSign}$. On input $m \in \{0,1\}^*$, vk as well as bd, the backdoor signing algorithm returns a signature via $\sigma \leftarrow\!\!\$\ \mathsf{BS.BSign}(\pi, bd, vk, m)$.

We define high effectiveness for BS as the amalgam of *high utility* and *high exclusivity*. High

Game $\mathbf{G}^{\text{ffe}}_{\mathsf{BS},n}$

INIT:
1 $(\pi, bd) \leftarrow\!\!\$\ \mathsf{BS.BPg}$; $\mathcal{E} \leftarrow \emptyset$
2 For $i = 1, 2, \ldots, n$ do
3 $\quad (vk_i, sk_i) \leftarrow\!\!\$\ \mathsf{BS.Kg}(\pi)$
4 Return $(\pi, vk_1, \ldots, vk_n)$

SIGN(i, m):
5 $\sigma \leftarrow\!\!\$\ \mathsf{BS.Sign}(\pi, sk_i, vk_i, m)$
6 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(i, m, \sigma)\}$; Return σ

BSIGN(i, m):
7 $\sigma \leftarrow\!\!\$\ \mathsf{BS.BSign}(\pi, bd, vk_i, m)$
8 $\mathcal{E} \leftarrow \mathcal{E} \cup \{(i, m, \sigma)\}$; Return σ

FIN(i, m, σ):
9 If $(i, m, \sigma) \in \mathcal{E}$ then return 0
10 Return $\mathsf{BS.Verify}(\pi, vk_i, m, \sigma)$

Game $\mathbf{G}^{\text{ind}}_{\mathsf{BS},n}$

INIT:
1 $(\pi, bd) \leftarrow\!\!\$\ \mathsf{BS.BPg}$; $b \leftarrow\!\!\$\ \{0, 1\}$
2 For $i = 1, 2, \ldots, n$ do
3 $\quad (vk_i, sk_i) \leftarrow\!\!\$\ \mathsf{BS.Kg}(\pi)$
4 Return $(\pi, vk_1, \ldots, vk_n)$

CHALL(i, m):
5 $\sigma_0 \leftarrow\!\!\$\ \mathsf{BS.Sign}(\pi, sk_i, vk_i, m)$
6 $\sigma_1 \leftarrow\!\!\$\ \mathsf{BS.BSign}(\pi, bd, vk_i, m)$
7 Return σ_b

SIGN(i, m):
8 $\sigma \leftarrow\!\!\$\ \mathsf{BS.Sign}(\pi, sk_i, vk_i, m)$
9 Return σ

BSIGN(i, m):
10 $\sigma \leftarrow\!\!\$\ \mathsf{BS.BSign}(\pi, bd, vk_i, m)$
11 Return σ

FIN(b'):
12 Return $(b = b')$

Figure 15: Games defining forgery-finding exclusivity (left) and indistinguishability (right) over n users.

utility asks that for every $(\pi, bd) \leftarrow\!\!\$\ \mathsf{BS.BPg}$, $(vk, sk) \leftarrow\!\!\$\ \mathsf{BS.Kg}(\pi)$ and $m \in \{0,1\}^*$, we have $\mathsf{BS.Verify}(\pi, vk, m, \mathsf{BS.BSign}(\pi, bd, vk, m)) = 1$. High exclusivity means that BS remains unforgeable to anyone without bd, even if they see signatures created using the backdoor. This is captured by game $\mathbf{G}^{\text{ffe}}_{\mathsf{BS},n}$ on the left of Figure 15, where n is the number of users and "ffe" denotes forgery-finding exclusivity. If A is an adversary, we let $\mathbf{Adv}^{\text{ffe}}_{\mathsf{BS},n}(A) = \Pr\left[\mathbf{G}^{\text{ffe}}_{\mathsf{BS},n}(A)\right]$ be its ffe advantage. Note that the difference from standard suf security is that A can also ask for signatures created using the backdoor via oracle BSIGN. An adversary A wins game $\mathbf{G}^{\text{ffe}}_{\mathsf{BS},n}$ if it produces a valid pair of message and signature for some user which was not generated by the SIGN or BSIGN oracle.

SINGLE-USER $\not\Rightarrow$ MULTI-USER. We define the game in a multi-user setting. In contrast to the unforgeability of a normal signature scheme, unforgeability for backdoored signature schemes 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 backdoor depends on the target verification key for which the signature is forged. A standard guessing argument will fail because the reduction will not be able to simulate signatures created by the backdoor for verification keys other than the challenge one. Below we will give a concrete counterexample separating the single-user from the multi-user setting. It essentially exploits that if the backdoor signing algorithm does not depend on the target verification key, a forged signature will verify under different (if not all) possible verification keys.

INDISTINGUISHABILITY. We define an additional property for BS, namely indistinguishability of honest and forged signatures. We define this via the multi-user game $\mathbf{G}^{\text{ind}}_{\mathsf{BS},n}$ on the right of Figure 15. For an adversary A, we let $\mathbf{Adv}^{\text{ind}}_{\mathsf{BS},n}(A) = 2\Pr\left[\mathbf{G}^{\text{ind}}_{\mathsf{BS},n}(A)\right] - 1$ be its ind advantage. Note that this is different from undetectability. While it may be obvious that a scheme contains a backdoor, this

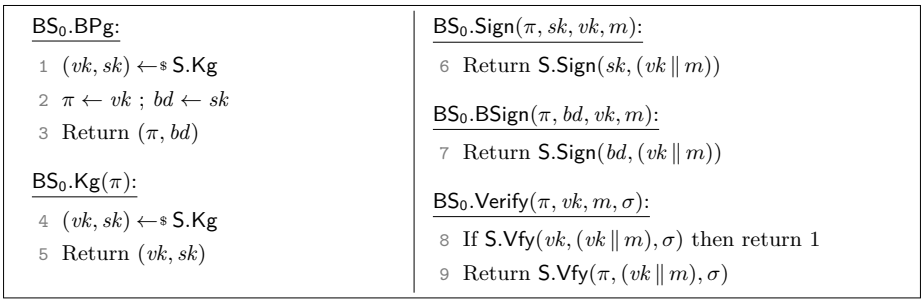

Figure 16: Construction of backdoored signature scheme BS_0 from a signature scheme S.

property ensures that a signature does not leak whether the backdoor has actually been used.

Related notions. The above notion is similar to identity-based signatures, except that user keys are generated only from π, not using the secret "master" key bd. Interestingly, there are already definitions of registration-based encryption, which has a similar end: to remove reliance on the master secret for user key-generation in IBE. Some examples are [25, 27]. In the former, user keys are generated only from the security parameter. In the latter, a CRS is allowed. In both, updatable public parameters are used in encryption and decryption. There does not seem to be a notion of registration-based signatures, however.

While our aim is to define highly-effective backdoored signatures, different categories of cryptographic subversion have been studied, such as subversion-resilient signatures [3] or anamorphic signatures [36]. Their goals are, however, quite different, as outlined in Section 3.

Naive construction. We construct a backdoored signature scheme BS_0 from a standard signature scheme in Figure 16. The idea is simple: The public parameter consists of a verification key and the backdoor is the corresponding signing key. The verification algorithm will accept signatures produced by either the backdoor signing key or the signing key of a user. Utility of the backdoor thus follows directly from the correctness of the signature scheme. Turning to exclusivity, we need to modify the signing algorithm to also include the user's verification key. More specifically, both signing algorithms create a signature on the concatenation of the user verification key and the actual message. This is necessary to achieve multi-user forgery-finding exclusivity. Note that without the verification key, a signature that was created using the backdoor verifies under all possible verification keys. We capture exclusivity of BS_0 in Theorem 7.1.

Theorem 7.1 *Let S be a signature scheme. From S we construct the backdoored signature scheme BS_0 as in Figure 16. Let n be a positive integer and let B be an adversary in game $\mathbf{G}^{\mathrm{ffe}}_{\mathsf{BS}_0,n}$. Then we can construct adversary A in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{S}}$ such that*

$$\mathbf{Adv}^{\mathrm{ffe}}_{\mathsf{BS}_0,n}(B) \leq (n+1) \cdot \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{S}}(A)\ . \tag{8}$$

If B makes q signing queries, then A makes at most q signing queries as well. Further, adversary A has running time close to that of B.

Proof: To show Eq. (8), note that the game $\mathbf{G}^{\mathrm{ffe}}_{\mathsf{BS}_0,n}$ describes an $(n+1)$-user game for standard suf-cma security. The only difference is that in order to sign, the verification keys are first concatenated with the message. The theorem thus follows directly from a simple guessing argument, as first shown in [24]. ▮

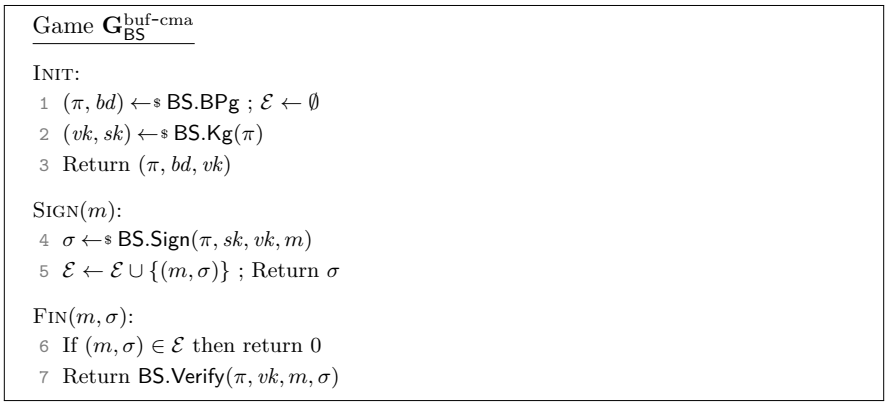

Figure 17: Game defining backdoor uf-cma of a backdoored signature scheme.

Adding indistinguishability. The above scheme does not satisfy indistinguishability because verification will only work for one of the verification keys. In order to achieve indistinguishability, we can add a non-interactive zero-knowledge proof. Whenever a signature is computed (either using the backdoor or the user signing key), one would commit to the signature and then prove in zero-knowledge that the signature verifies. Verification then simply verifies that proof.

Existing schemes with backdoors. One might wonder whether it is possible to show for existing signature schemes that they contain such a highly-effective backdoor. Often signature schemes specify a trusted setup. One example is the signature scheme based on the identification scheme of Guillou and Quisquater [29]. In their scheme, the public parameters consist of an RSA modulus. Knowing the factorization indeed allows one to compute arbirtrary signatures which are identically distributed to honestly generated signatures. Thus, the scheme can be viewed as a backdoored scheme that satisfies utility, forgery-finding exclusivity and indistinguishability.

8 Positive results for signatures

In this section we provide positive results for existing schemes where the setup suggests that the scheme could potentially be backdoored. Our goal is then to show that this potential backdoor is not effective. To capture this, we define an additional game, namely backdoor unforgeability under chosen message attack (buf-cma). The game is given in Figure 17 and is essentially the suf-cma game for signatures, but the adversary is additionally given the backdoor.

Okamoto signatures. The Okamoto signature scheme [39] is based on a prime-order group $(\mathbb{G}, p, g)$. The parameter generation chooses an additional group element h at random. We may ask (as was the case for Dual EC) whether the discrete logarithm α of h to base g can be used as a backdoor to the scheme; that is, to forge signatures. We denote the Okamoto signature scheme which retains the discrete logarithm by BOS and give its full description in Figure 18. It is worth noting that existing security proofs rely on solving the discrete logarithm for h. Maybe somewhat surprisingly, we are able to provide an alternative proof showing buf-cma for BOS relying on the security of the Schnorr signature scheme [43], which we recall in Figure 19.

Theorem 8.1 *Let* BOS *and* SS *be the backdoored Okamoto signature scheme and the Schnorr signature scheme, respectively. Let* B *be an adversary in game* $\mathbf{G}^{\text{buf-cma}}_{\mathsf{BOS}}$*. Then we can construct*

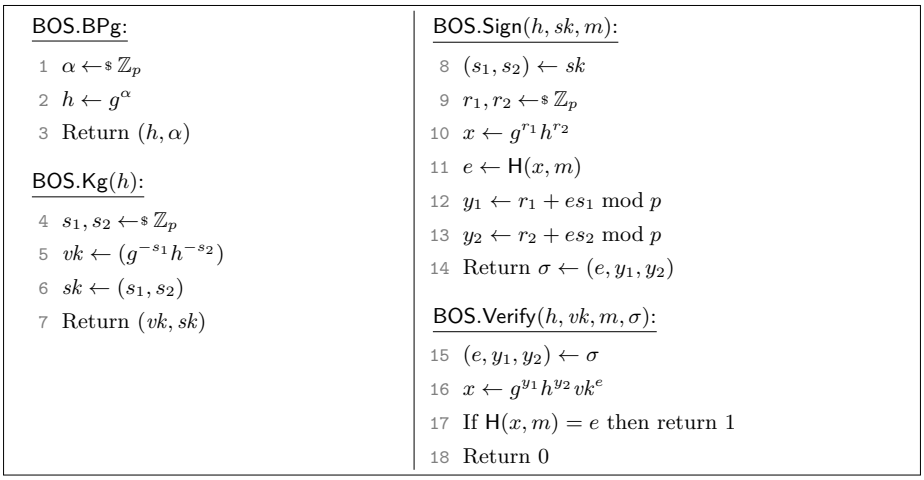

Figure 18: Backdoored Okamoto signature scheme BOS for a prime-order group $(\mathbb{G}, p, g)$ and hash function H. The parameter generation picks h in a way such that the discrete logarithm α is known.

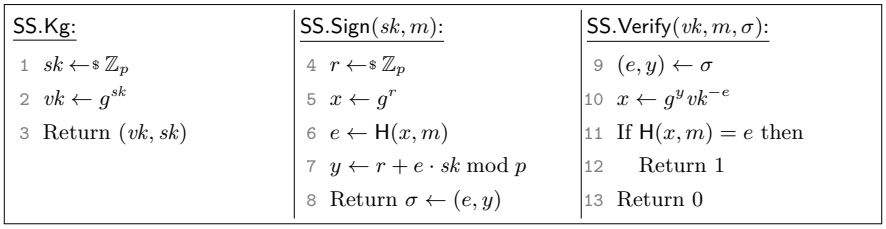

Figure 19: Schnorr scheme SS for a prime-order group $(\mathbb{G}, p, g)$ and hash function H.

adversary A in game $\mathbf{G}^{\text{suf-cma}}_{\mathsf{SS}}$ such that

$$\mathbf{Adv}^{\text{buf-cma}}_{\mathsf{BOS}}(B) \leq \mathbf{Adv}^{\text{suf-cma}}_{\mathsf{SS}}(A) \; . \tag{9}$$

Adversary A makes the same number of signing queries as B. Further, A has running time close to that of B.

Proof: We construct adversary A as shown in Figure 20. A gets as input the verification key $vk = g^{sk}$ for SS. It picks α to compute h and runs adversary B on (h, vk', α), where $vk' = vk^{-1} = g^{-sk}$. When B issues a signing query on m, A forwards this query to its own oracle to receive a signature (e, y), where $e = \mathsf{H}(x, m)$ for $x = g^r$ and $y = r + e \cdot sk$. A now picks y_2 at random and computes $y_1 = y - \alpha y_2$. Note that this is a valid Okamoto signature since

$$g^{y_1} \cdot h^{y_2} \cdot (vk')^{e} = g^{y - \alpha y_2} \cdot g^{\alpha y_2} \cdot g^{-sk \cdot e} = g^{y - e \cdot sk} = g^{y} \cdot vk^{-e} = x \; .$$

When B terminates and outputs a forgery $(m, (e, y_1, y_2))$, A computes $y = y_1 + \alpha \cdot y_2$ and outputs $(m, (e, y))$ as its own forgery. If B's forgery is valid, then

$$g^{y_1} \cdot h^{y_2} \cdot (vk')^{e} = g^{y_1 + \alpha y_2} \cdot vk^{-e} = g^{y} \cdot vk^{-e} = x \; .$$

Hence, A outputs a valid forgery as well. ∎

Remark 8.2 *For better concrete security bounds, the verification key is often included inside the hash. In this case, the above proof still holds by programming the random oracle accordingly.*

Adversary $A(vk)$:	Oracle $\textsc{Sign}(m)$:
1 $\alpha \leftarrow\!\!\$\ \mathbb{Z}_p$; $h \leftarrow g^\alpha$	7 $\sigma \leftarrow \mathbf{G}_{\mathsf{SS}}^{\text{suf-cma}}.\textsc{Sign}(m)$
2 $vk' \leftarrow vk^{-1}$	8 $(e, y) \leftarrow \sigma$
3 $(m, \sigma) \leftarrow B[\textsc{Sign}](h, vk', \alpha)$	9 $y_2 \leftarrow\!\!\$\ \mathbb{Z}_p$; $y_1 \leftarrow y - \alpha \cdot y_2 \bmod p$
4 $(e, y_1, y_2) \leftarrow \sigma$	10 $\sigma' \leftarrow (e, y_1, y_2)$
5 $y \leftarrow y_1 + \alpha \cdot y_2 \bmod p$	11 Return σ'
6 Return $(m, (e, y))$	

Figure 20: Adversary A for the proof of Theorem 8.1.

However, this introduces an additional statistical term to the bound which accounts for the event that the random oracle has already been queried on a value x that is later used for a signature. Since x is distributed uniformly, the probability that this bad event happens is negligible.

KATZ-WANG SIGNATURES. A similar result can be proven for the Katz-Wang signature scheme [34]. In Katz-Wang, the public parameters also include a group element h and the verification key is a tuple (g^{sk}, h^{sk}). Signing works as in Schnorr, except that it computes (g^r, h^r) from which e is computed as $\mathsf{H}(g^r, h^r, m)$. The signature is then the tuple (e, y), where $y = r + sk \cdot e$. Thus, a reduction can simulate signatures using its own signing oracle and programming the random oracle accordingly.

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 6.)

[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 2, 3, 4, 5, 7, 15.)

[3] 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 24.)

[4] 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 2, 7.)

[5] F. Banfi, K. Gegier, M. Hirt, and U. Maurer. Anamorphic encryption, revisited. Cryptology ePrint Archive, Report 2023/249, 2023. https://eprint.iacr.org/2023/249. (Cited on 2, 6.)

[6] 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 2, 7.)

[7] 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, 6.)

[8] 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 8, 13, 36.)

[9] 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 2, 6.)

[10] 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 22, 38.)

[11] 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 9.)

[12] S. Checkoway, J. Maskiewicz, C. Garman, J. Fried, S. Cohney, M. Green, N. Heninger, R. Weinmann, E. Rescorla, and H. Shacham. Where did I leave my keys?: lessons from the juniper dual EC incident. *Commun. ACM*, 61(11):148–155, 2018. (Cited on 5.)

[13] S. Checkoway, R. Niederhagen, A. Everspaugh, M. Green, T. Lange, T. Ristenpart, D. J. Bernstein, J. Maskiewicz, H. Shacham, and M. Fredrikson. On the practical exploitability of dual EC in TLS implementations. In K. Fu and J. Jung, editors, *USENIX Security 2014*, pages 319–335. USENIX Association, Aug. 2014. (Cited on 2, 5.)

[14] 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 3.)

[15] 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 20.)

[16] 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 9.)

[17] 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 2, 7, 22.)

[18] 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 4, 5, 7, 15, 18, 19, 34.)

[19] 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 2, 5, 7, 22.)

[20] 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 7.)

[21] 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 6.)

[22] 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, 3, 4, 5, 7, 10, 15, 18, 22.)

[23] A. Freund. Openwall oss-security mailing list, 29 Mar. 2024. https://www.openwall.com/lists/oss-security/2024/03/29/4. (Cited on 6, 22.)

[24] S. Galbraith, J. Malone-Lee, and N. Smart. Public key signatures in the multi-user setting. *Information Processing Letters*, 83(5):263–266, 2002. (Cited on 24.)

[25] S. Garg, M. Hajiabadi, M. Mahmoody, and A. Rahimi. Registration-based encryption: Removing private-key generator from IBE. In A. Beimel and S. Dziembowski, editors, *TCC 2018, Part I*, volume 11239 of *LNCS*, pages 689–718. Springer, Heidelberg, Nov. 2018. (Cited on 24.)

[26] 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 9.)

[27] N. Glaeser, D. Kolonelos, G. Malavolta, and A. Rahimi. Efficient registration-based encryption. In *ACM CCS 2023*, pages 1065–1079. ACM Press, Nov. 2023. (Cited on 24.)

[28] O. Goldreich. *Foundations of Cryptography: Basic Applications*, volume 2. Cambridge University Press, Cambridge, UK, 2004. (Cited on 9.)

[29] L. C. Guillou and J.-J. Quisquater. A practical zero-knowledge protocol fitted to security microprocessor minimizing both trasmission and memory. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 123–128. Springer, Heidelberg, May 1988. (Cited on 25.)

[30] D. Hofheinz and E. Kiltz. Programmable hash functions and their applications. *Journal of Cryptology*, 25(3):484–527, July 2012. (Cited on 7.)

[31] 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 2, 6.)

[32] D. Johnson, A. Menezes, and S. Vanstone. The elliptic curve digital signature algorithm (ECDSA). *International Journal of Information Security*, 2001. (Cited on 34.)

[33] B. Kaliski. PKCS #5: Password-based cryptography specification. RSA Laboratories, Sept. 2000. Version 2.0. (Cited on 3.)

[34] J. Katz and N. Wang. Efficiency improvements for signature schemes with tight security reductions. In S. Jajodia, V. Atluri, and T. Jaeger, editors, *ACM CCS 2003*, pages 155–164. ACM Press, Oct. 2003. (Cited on 5, 6, 27.)

[35] H. Krawczyk and T. Rabin. Chameleon signatures. In *NDSS 2000*. The Internet Society, Feb. 2000. (Cited on 5, 7, 18.)

[36] 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 2, 6, 24.)

[37] 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 21.)

[38] 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 21.)

[39] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In E. F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 31–53. Springer, Heidelberg, Aug. 1993. (Cited on 5, 6, 25.)

[40] OpenSSL. https://www.openssl.org/. (Cited on 38.)

[41] 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 2, 6.)

[42] 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 7.)

[43] C.-P. Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 239–252. Springer, Heidelberg, Aug. 1990. (Cited on 25.)

[44] C.-P. Schnorr. Efficient signature generation by smart cards. *Journal of Cryptology*, 4(3):161–174, Jan. 1991. (Cited on 6.)

[45] 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 2, 6.)

[46] 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 2, 6.)

[47] 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 2, 6.)

[48] 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 3, 7, 21.)

[49] 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 3, 7, 21.)

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

[51] 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 21.)

[52] 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, 6.)

[53] 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, 6.)

[54] 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 2, 6.)

[55] G. Zaverucha and D. Shumow. Are certificate thumbprints unique? Cryptology ePrint Archive, Report 2019/130, 2019. https://eprint.iacr.org/2019/130. (Cited on 3, 7, 21.)

A Proof of $\mathsf{BH} \implies$ FJM

Proof of Proposition 5.2: We first consider finding a preimage. Adversary B_1 is given in Figure 21. On inputs $\mathit{fbd}, \mathit{fhk}, y$, B_1 simply runs $\mathsf{BH.FP}$ to find a preimage of y. If BH achieves high utility for P then $m' \leftarrow\!\!\$\ \mathsf{BH.FP}(\mathit{fbd}, \varepsilon, y)$ is such that $\mathsf{BH.Ev}(\mathit{fhk}, m') = \mathsf{FH.Ev}(\mathit{fhk}, m') = y$ and $\mathsf{P}(m', \varepsilon) = \mathsf{true}$. The former is all that is needed to be a successful preimage in game $\mathbf{G}^{\text{f-pr}}_{\mathsf{FH},1}$ so $\mathbf{Adv}^{\text{f-pr}}_{\mathsf{FH},1}(B_1) = 1$.

Next we aim to find a second preimage and make the additional assumption on P in the proposition statement. Adversary B_2 is in Figure 21. On inputs $\mathit{fbd}, \mathit{fhk}, y, m$, B_2 runs $\mathsf{BH.FP}$ to find a preimage of y, subject to $u = m$. If BH achieves high utility for P then $m' \leftarrow\!\!\$\ \mathsf{BH.FP}(\mathit{fbd}, m, y)$ is such that $\mathsf{BH.Ev}(\mathit{fhk}, m') = \mathsf{FH.Ev}(\mathit{fhk}, m') = y$ and $\mathsf{P}(m', m) = \mathsf{true}$. Our restriction on P then means that $m' \neq m$, so both winning conditions of game $\mathbf{G}^{\text{f-2pr}}_{\mathsf{FH},1}$ are met. Thus $\mathbf{Adv}^{\text{f-2pr}}_{\mathsf{FH},1}(B_2) = 1$. ▌

Proof of Theorem 5.3: Given adversary B, A operates according to the pseudocode in Figure 21. By inspection, B is precisely in game $\mathbf{G}^{\text{f-cr}}_{\mathsf{FH},0}$ instantiated with FH of Figure 10. If B returns a successful collision (m_1, m_2) then $m_1 \neq m_2$ and $\mathsf{FH.Ev}(\mathit{fhk}, m_1) = \mathsf{FH.Ev}(\mathit{fhk}, m_2)$. Since $\mathsf{FH.Ev}(\mathit{fhk}, m) = \mathsf{BH.Ev}(\mathit{hk}, m)$ for this construction, (m_1, m_2) remains a successful collision in game $\mathbf{G}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}$. This proves Eq. (6) and the theorem. ▌

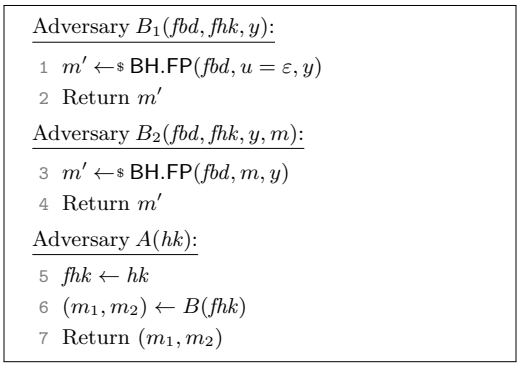

Figure 21: Adversaries B_1, B_2 for the proof of Proposition 5.2, and adversary A for the proof of Theorem 5.3.

B Proof of enhanced collision resistance

Proof of Proposition 5.4: To prove the correctness of CH, let $(h, r) \leftarrow\!\!\$\ \mathsf{CH.Hash}(pk, m)$. Then (h, r) is chosen via $r \leftarrow\!\!\$\ \mathsf{Emb.ES}$ and $h \leftarrow \mathsf{BH.Ev}(pk, \mathsf{Emb}(r, m))$. Correctness now asks that $\mathsf{CH.Check}(pk, m, h, r) = 1$, or for this construction that $h = \mathsf{BH.Ev}(pk, \mathsf{Emb}(r, m))$. This is satisfied because $\mathsf{BH.Ev}$ and Emb are deterministic. Correctness also considers the outputs of $\mathsf{CH.Adapt}$. For this, let $m' \in \mathsf{CH.Dom}$ and $r' \leftarrow\!\!\$\ \mathsf{CH.Adapt}(sk, m, m', r, h)$. That is, for $e' \leftarrow\!\!\$\ \mathsf{BH.FP}(sk, m', h)$, r' is such that $(r', u) \leftarrow \mathsf{Emb}^{-1}(e')$. Let us now consider $\mathsf{CH.Check}(pk, m', r', h)$. This returns 1 iff $h = \mathsf{BH.Ev}(pk, \mathsf{Emb}(r', m'))$. If BH achieves high utility for P, then e' is found such that $\mathsf{P}(e', m') = 1$ and $\mathsf{BH.Ev}(pk, e') = h$.

We next claim that $\mathsf{Emb}(r', m') = e'$, which in combination with the above will complete the proof that $\mathsf{CH.Check}(pk, m', r', h) = 1$. This is where we use the assumption that $(\mathsf{P}(e, u) = \mathsf{true}) \implies (\mathsf{Emb}^{-1}(e) = (x, u))$ for some $x \in \mathsf{Emb.ES}$. In our case, $\mathsf{P}(e', m') = 1$ implies that $\mathsf{Emb}^{-1}(e') = (x, m')$ for some $x \in \mathsf{Emb.ES}$. As on line 6 of Figure 12, we in fact have $x = r'$ and $u = m'$. That is, $\mathsf{Emb}^{-1}(e') = (r', m')$ and thus $\mathsf{Emb}(r', m') = e'$.

We have proved that CH is correct but let us briefly justify the additional assumption on P and Emb. This is in particular satisfied by $\mathsf{P}^n_{\mathsf{sfx}}$ and $\mathsf{Emb}^n_{\mathsf{sfx}}$ of Figure 4, and potentially by a larger class of possible predicates. Assume that the boxed code in Figure 4 is included. Recall that $\mathsf{Emb}^n_{\mathsf{sfx}}.\mathsf{ES} = \{0,1\}^n$, $\mathsf{Emb}^n_{\mathsf{sfx}}(x, u) = x \,\|\, u$, and $\mathsf{EmbInv}^n_{\mathsf{sfx}}(e) = (x, u)$ where x is the first n bits of e, and u is the remaining suffix (of unspecified length). Now observe that if $\mathsf{P}^n_{\mathsf{sfx}}(e, u) = \mathsf{true}$ then u is a suffix of e such that $|e| = |u| + n$, where the latter is the condition in the boxed code. Then $e = x \,\|\, u$ where $|x| = n$ so $\mathsf{Emb}^{-1}(e) = (x, u)$, which is the desired property. ▌

Proof of Theorem 5.5: Consider game G_0 of Figure 22. We claim that

$$\mathbf{Adv}^{\text{ch-ecr}}_{\mathsf{CH}}(B) = \Pr[\,\mathrm{G}_0(B)\,]\,. \tag{10}$$

Eq. (10) is justified by the fact that G_0 is the same as game $\mathbf{G}^{\text{ch-ecr}}_{\mathsf{CH}}$ when instantiated with scheme CH of Figure 12. The set $\mathcal{E}$ is additionally accounted for on lines 2,8 but all oracle return values are as in $\mathbf{G}^{\text{ch-ecr}}_{\mathsf{CH}}$.

For the remainder of the proof, we turn to designing an adversary A such that

$$\Pr[\,\mathrm{G}_0(B)\,] \le \mathbf{Adv}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}(A)\,. \tag{11}$$

Game G_0

INIT:
1 $(hk, bd) \leftarrow\!\!\$\ \mathsf{BH.BKg}$; $pk \leftarrow hk$; $sk \leftarrow bd$
2 $\mathcal{E} \leftarrow \emptyset$; $\mathcal{H} \leftarrow \emptyset$
3 Return pk

ADAPT(m, m', r, h):
4 If $(h \neq \mathsf{BH.Ev}(pk, \mathsf{Emb}(r, m)))$ then return $\perp$
5 $e' \leftarrow\!\!\$\ \mathsf{BH.FP}(sk, m', h)$
6 $(r', u) \leftarrow \mathsf{Emb}^{-1}(e')$ // where $(u = m')$
7 If $(r' = \perp)$ then return $\perp$
8 $\mathcal{E} \leftarrow \mathcal{E} \cup \{e'\}$; $\mathcal{H} \leftarrow \mathcal{H} \cup \{h\}$
9 Return r'

FIN(m_1, m_2, r_1, r_2, h):
10 $b \leftarrow ((m_1 \neq m_2) \wedge \mathsf{CH.Check}(pk, m_1, r_1, h) \wedge \mathsf{CH.Check}(pk, m_2, r_2, h))$
11 Return $(b \wedge (h \notin \mathcal{H}))$

Figure 22: Game G_0 for the proof of Theorem 5.5. On line 6, we have $u = m'$ which is guaranteed by the assumption on P, Emb in the theorem statement; this was proved in the proof of Proposition 5.4.

Adversary $A(hk)$:
1 $pk \leftarrow hk$
2 $(m_1, m_2, r_1, r_2, h) \leftarrow B[\mathrm{ADAPT}_A](pk)$
3 $e_1 \leftarrow \mathsf{Emb}(r_1, m_1)$; $e_2 \leftarrow \mathsf{Emb}(r_2, m_2)$
4 Return (e_1, e_2)

Oracle $\mathrm{ADAPT}_A(m, m', r, h)$:
5 If $(h \neq \mathsf{BH.Ev}(pk, \mathsf{Emb}(r, m)))$ then return $\perp$
6 $e' \leftarrow \mathrm{GETPMG}(m', h)$
7 $(r', u) \leftarrow \mathsf{Emb}^{-1}(e')$
8 Return r'

Figure 23: Adversary A for the proof of Theorem 5.5.

Let adversary A operate according to the pseudocode in Figure 23. Eq. (11) requires two explanations. First, the input pk and oracle ADAPT_A that are given to B in Figure 23 are the same as in game G_0. Second, $[\mathrm{G}_0(B) \text{ outputs } \mathsf{true}] \implies [\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}(A) \text{ outputs } \mathsf{true}]$. To justify the first, note that pk given to B is a properly generated public key of $\mathsf{BH.BKg}$ and thus of $\mathsf{CH.Kg}$. The oracle ADAPT_A performs the same steps as oracle ADAPT of Figure 22, where the call to GETPMG on line 6 of ADAPT_A implements the call to $\mathsf{BH.FP}$ on line 5 of ADAPT. Also note that A makes one query to GETPMG for each of B's queries to ADAPT_A, proving the running time of A in the theorem statement.

Finally we turn to proving that

$$[\mathrm{G}_0(B) \text{ outputs } \mathsf{true}] \implies [\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}(A) \text{ outputs } \mathsf{true}] \,. \tag{12}$$

This requires consideration of the three conditions in the FIN oracles of games G_0 and $\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}$. In particular, suppose B's final output is (m_1, m_2, r_1, r_2, h) so that A outputs (e_1, e_2) where $e_i \leftarrow \mathsf{Emb}(r_i, m_i)$ for $i \in \{1, 2\}$. We claim that the following three statements hold, where the lefthand

statement is in game $\mathrm{G}_0(B)$ and the righthand statement is in $\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}(A)$:

$$(m_1 \neq m_2) \implies (e_1 \neq e_2) \tag{13}$$

$$(h = \mathsf{BH.Ev}(pk, \mathsf{Emb}(r_1, m_1)) = \mathsf{BH.Ev}(pk, \mathsf{Emb}(r_2, m_2))) \implies (\mathsf{BH.Ev}(pk, e_1) = \mathsf{BH.Ev}(pk, e_2)) \tag{14}$$

$$(h \notin \mathcal{H}) \implies ((e_1 \notin \mathcal{E}) \wedge (e_2 \notin \mathcal{E})) \ . \tag{15}$$

If all three of the above hold then Eq. (12) will follow. To start, Eq. (13) holds because Emb is invertible via Emb^{-1}. Specifically, if $e_1 = \mathsf{Emb}(r_1, m_1)$ then $\mathsf{Emb}^{-1}(e_1) = (r_1, m_1)$. This holds for e_2 as well, which means that if $m_1 \neq m_2$ then we must have $\mathsf{Emb}(r_1, m_1) \neq \mathsf{Emb}(r_2, m_2)$. Eq. (14) has a similarly simple justification: both Emb and $\mathsf{BH.Ev}$ are deterministic.

Let us now consider Eq. (15), and suppose that the lefthand statements of Eqs. (13),(14) already hold. We now prove Eq. (15) by contrapositive; suppose that $e_i \in \mathcal{E}$ for at least one $i \in \{1, 2\}$. Looking at game G_0, $\mathcal{E}$ is updated on lines 5,8. If $e_i \in \mathcal{E}$ then $e_i \leftarrow \mathsf{BH.FP}(sk, m', h')$ was computed on line 5 for some m', h' during a query $\textsc{Adapt}(m^*, m', r^*, h')$. Correctness of $\mathsf{BH.FP}$ guarantees that $h' = h$. Thus h was added to $\mathcal{H}$ on line 8. This contradiction proves Eq. (15) and completes the proof. ∎

C Proof of full collision resistance

In Section 5 we constructed a chameleon hash function CH given backdoored hash function BH and embedding function Emb. Theorem 5.5 proved that CH satisfies enhanced collision resistance if BH satisfies cfe exclusivity. However, "enhanced" is not the strongest notion of cr security for chameleon hash functions; currently that is full collision resistance as defined by [18] and given in Figure 11. We can prove that our constructed CH satisfies full cr if an additional assumption is made on the min-entropy of $\mathsf{BH.FP}$. For example, in our construction $\mathsf{BH} = \mathbf{HEB}[\mathsf{G}, \mathsf{S}, \mathsf{Emb}]$ of Section 4, the randomization of $\mathsf{BH.FP}$ is coming from the randomization of $\mathsf{S.Sign}$.

The specific definition we use is the following. For BH and embedding function Emb, let $\mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}$ denote, over choices of m', r, h and $(hk, bd) \leftarrow\!\!\$\ \mathsf{BH.BKg}$, the maximum probability that $(r' = r)$ when r' is computed according to experiment: $e' \leftarrow\!\!\$\ \mathsf{BH.FP}(bd, m', h)$; $(r', u) \leftarrow \mathsf{Emb}^{-1}(e')$; return r'. Let $\mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}(q)$ denote, over q choices of m'_i, r_i, h_i and fixed $(hk, bd) \leftarrow\!\!\$\ \mathsf{BH.BKg}$, the probability that $(r'_i = r_i)$ in at least one of these q experiments with m'_i, r_i, h_i and bd. This is not precisely the min-entropy of $\mathsf{BH.FP}$, as there is an additional Emb^{-1} application; nonetheless it is intuitively close to the min-entropy of $\mathsf{BH.FP}$.

The value $\mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}(q)$ depends on the randomness of $\mathsf{BH.FP}$. As one example, suppose $\mathsf{BH} = \mathbf{HEB}[\mathsf{G}, \mathsf{S}, \mathsf{Emb}]$ is instantiated with a deterministic signature scheme. Then there is only one possible value of e' for any bd, m', h and thus one possible r' in the above experiment. Selecting $m', r = r', h$ shows that $\mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}(q) = 1$. On the other hand, suppose a randomized signature scheme is used. In particular, consider ECDSA [32] where the nonce $N \in \{0,1\}^k$ used during signing is cryptographically random. An embedding value r' is randomly selected to equal a chosen r only if a particular signature, and thus particular nonce, is chosen. Over q queries, $\mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}(q) \approx \frac{q^2}{2^k}$, where 2^k is approximately the ECDSA group order; for example $2^k \approx 2^{256}$.

In the below theorem, the term $\mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}(q)$ is not an artifact of looseness; there is a natural setting where the reduction fails. Consider B which aims to produce CHF collision (m_1, m_2, r_1, r_2, h). B can then make a query $r_3 \leftarrow\!\!\$\ \textsc{Adapt}(m_1, m_3, r_1, h)$ and $r'_1 \leftarrow\!\!\$\ \textsc{Adapt}(m_3, m_1, r_3, h)$. With some probability $r'_1 = r_1$; this depends on the randomness of $\mathsf{BH.FP}$, followed by Emb^{-1}. If this occurs,

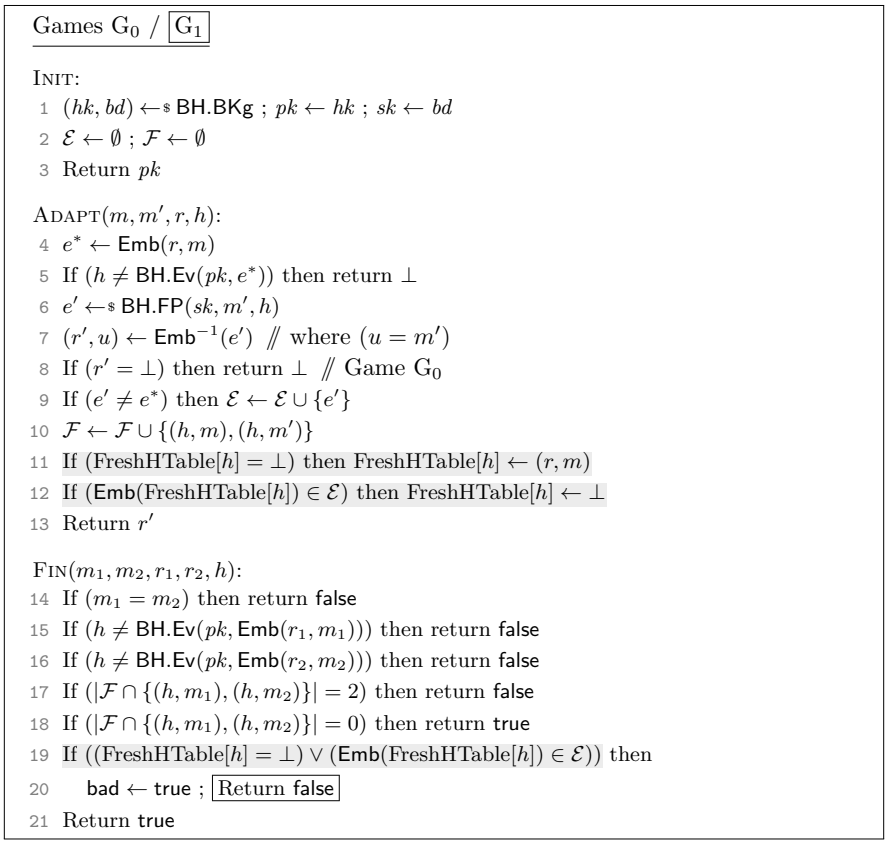

Figure 24: Games G_0, G_1 for the proof of Theorem C.1. On line 7, we have $u = m'$ which is guaranteed by the assumption on P, Emb in the theorem statement; this was proved in the proof of Proposition 5.4. G_1 includes the boxed code while G_0 does not.

B still wins because $(h, m_2) \notin \mathcal{F}$. However, the constructed A cannot use either $\mathsf{Emb}(r_1, m_1)$ or $\mathsf{Emb}(r_3, m_3)$ in a collision because they are both in $\mathcal{E}$. There is no obvious way for A to create a collision in this setting, as B has made queries for the sole purpose of adding potentially useful preimages to $\mathcal{E}$.

Thus the below theorem is subtle, but the takeaway is that full cr of CH is achievable if the underlying scheme $\mathsf{BH} = \mathbf{HEB}[\mathsf{G}, \mathsf{S}, \mathsf{Emb}]$ uses S with sufficiently random signatures, or otherwise has sufficiently randomized $\mathsf{BH.FP}$.

Theorem C.1 *Let* BH *be a backdoored hash function. From* BH *we construct chameleon hash function* CH *as in Figure 12. Let* P *be a predicate with a correct embedding function* Emb *such that* $(\mathsf{P}(e, u) = \mathsf{true}) \implies (\mathsf{Emb}^{-1}(e) = (x, u))$ *for some* $x \in \mathsf{Emb.ES}$. *Suppose* BH *achieves high utility for* P. *Then if* B *is an adversary in game* $\mathbf{G}^{\text{ch-full}}_{\mathsf{CH}}$, *we can construct adversary* A *in game* $\mathbf{G}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}$ *such that*

$$\mathbf{Adv}^{\text{ch-full}}_{\mathsf{CH}}(B) \leq \mathbf{Adv}^{\text{cfe}}_{\mathsf{BH},\mathsf{P}}(A) + \mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}(q) \ , \tag{16}$$

where if B *makes* q ADAPT *queries then* A *makes* q GETCOLL *queries. Adversary* A *has running time close to that of* B.

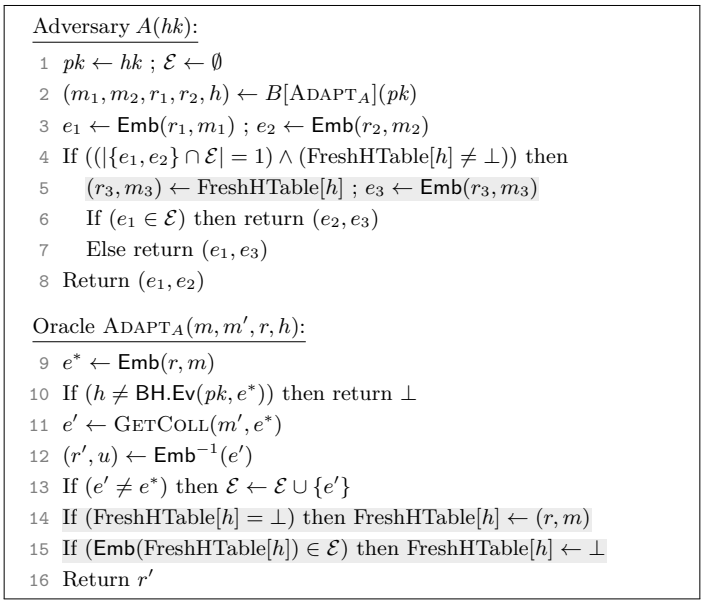

Figure 25: Adversary A for the proof of Theorem C.1.

Proof of Theorem C.1: Consider game G_0 of Figure 24. We claim that

$$\mathbf{Adv}^{\text{ch-full}}_{\mathsf{CH}}(B) = \Pr[\,\mathrm{G}_0(B)\,]\,. \tag{17}$$

Eq. (17) is justified by the fact that G_0 is the same as game $\mathbf{G}^{\text{ch-full}}_{\mathsf{CH}}$ when instantiated with scheme CH of Figure 12. The set $\mathcal{E}$ is additionally accounted for on lines 2,9 and the table FreshHTable is tracked on the highlighted lines, but these do not alter the ADAPT responses in G_0. The FIN procedure is split into several checks that are equivalent to those of $\mathbf{G}^{\text{ch-full}}_{\mathsf{CH}}$ (the boxed code is omitted in G_0).

Next we turn to G_1. Note that this excludes line 8 but correctness of $\mathsf{BH.FP}$ and the assumption on P, Emb in the theorem statement imply that it will never be the case that $(r' = \perp)$. Games $\mathrm{G}_0, \mathrm{G}_1$ thus only differ at line 20 and are identical-until-bad. The Fundamental Lemma of Game Playing [8] implies that

$$\Pr[\,\mathrm{G}_0(B)\,] - \Pr[\,\mathrm{G}_1(B)\,] \leq \Pr[\,\mathrm{G}_1(B) \text{ sets } \mathsf{bad}\,]$$
$$\Pr[\,\mathrm{G}_0(B)\,] \leq \Pr[\,\mathrm{G}_1(B)\,] + \Pr[\,\mathrm{G}_1(B) \text{ sets } \mathsf{bad}\,]\,.$$

To complete the proof of Eq. (16), we design an adversary A and prove that

$$\Pr[\,\mathrm{G}_1(B)\,] \leq \mathbf{Adv}^{\text{cfe}}_{\mathsf{BH,P}}(A) \tag{18}$$

$$\Pr[\,\mathrm{G}_1(B) \text{ sets } \mathsf{bad}\,] \leq \mathsf{R}^{-1}_{\mathsf{BH,Emb}}(q)\,. \tag{19}$$

We begin with Eq. (18). Let adversary A operate according to the pseudocode in Figure 25. First, the input pk and oracle ADAPT_A that are given to B in Figure 25 are the same as in game G_1. Second, $[\mathrm{G}_1(B) \text{ outputs } \mathsf{true}] \implies [\mathbf{G}^{\text{cfe}}_{\mathsf{BH,P}}(A) \text{ outputs } \mathsf{true}]$. To justify the first, note that pk given to B is a properly generated public key of $\mathsf{BH.BKg}$ and thus of $\mathsf{CH.Kg}$. The oracle ADAPT_A performs the same steps as oracle ADAPT of Figure 24, where the call to GETCOLL on line 11 of ADAPT_A implements the call to $\mathsf{BH.FP}$ on line 6 of ADAPT. Also note that A makes one query to GETCOLL for each of B's queries to ADAPT_A, proving the running time of A in the theorem

statement.

Next we turn to proving that

$$[\mathrm{G}_1(B) \text{ outputs } \mathsf{true}] \implies [\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}(A) \text{ outputs } \mathsf{true}] \,. \tag{20}$$

There are three conditions in the FIN oracles of games G_1 and $\mathbf{G}^{\mathrm{cfe}}_{\mathsf{BH},\mathsf{P}}$. Two of the conditions are simple; suppose B outputs (m_1, m_2, r_1, r_2, h) so that A sets $e_1 = \mathsf{Emb}(r_1, m_1), e_2 = \mathsf{Emb}(r_2, m_2)$. First, if $(m_1 \neq m_2)$ then $(e_1 \neq e_2)$ due to correctness of Emb^{-1}. Second, if $\mathsf{BH.Ev}(pk, \mathsf{Emb}(r_1, m_1)) = \mathsf{BH.Ev}(pk, \mathsf{Emb}(r_2, m_2))$ then $\mathsf{BH.Ev}(pk, e_1) = \mathsf{BH.Ev}(pk, e_2)$. For the final condition we must consider the sets $\mathcal{E}, \mathcal{F}$. There are two cases where $\mathrm{G}_1(B)$ outputs true. In the first, $|\mathcal{F} \cap \{(h, m_1), (h, m_2)\}| = 0$. We claim that if $e_i \in \mathcal{E}$ then $(h, m_i) \in \mathcal{F}$. This will imply that in this first case, we must have $|\mathcal{E} \cap \{e_1, e_2\}| = 0$ so that A's output (e_1, e_2) is a winning collision. To prove the claim, suppose $e_i \in \mathcal{E}$. Then e_i was computed on lines 6,7 by $\mathsf{BH.FP}$ and it must have been that $m' = m_i$ during that ADAPT query, due to correctness of $\mathsf{BH.FP}$ and the condition on Emb. Thus (h, m_i) was added to $\mathcal{F}$.

Let us now consider the second case in which $\mathrm{G}_1(B)$ outputs true, when $|\mathcal{F} \cap \{(h, m_1), (h, m_2)\}| = 1$. To analyze this second case, we will use the table FreshHTable. The entry at FreshHTable$[h]$ is always either $\perp$ or a tuple (r, m) such that $\mathsf{Emb}(r, m) \notin \mathcal{E}$ and $\mathsf{BH.Ev}(pk, \mathsf{Emb}(r, m)) = h$. These conditions are maintained by lines 5,11,12 of G_1. Now suppose execution of G_1 has reached line 19 of FIN. Since the boxed code is included in G_1, $\mathrm{G}_1(B)$ only outputs true at this point if FreshHTable$[h] \neq \perp$ and $\mathsf{Emb}(\text{FreshHTable}[h]) \notin \mathcal{E}$. Correspondingly, in this case adversary A sets $(r_3, m_3) \leftarrow \mathsf{Emb}(\text{FreshHTable}[h])$ and $e_3 \leftarrow \mathsf{Emb}(r_3, m_3)$. Suppose without loss of generality that $(h, m_1) \notin \mathcal{F}$ and $(h, m_2) \in \mathcal{F}$. Then m_1 has never been encountered during an ADAPT query so in particular $m_1 \neq m_3$. We also have that $e_1 \notin \mathcal{E}$. Thus A returns (e_1, e_3) which is a collision to hash h such that $e_1 \notin \mathcal{E}$ and $e_3 \notin \mathcal{E}$. This completes the justification of Eq. (18).

The above analysis shows that the sets $\mathcal{E}, \mathcal{F}$ must be analyzed carefully. To prove Eq. (19) we now consider the bad flag of game G_1. This is set if line 19 of FIN is reached such that FreshHTable$[h] = \perp$ or $\mathsf{Emb}(\text{FreshHTable}[h]) \in \mathcal{E}$. The latter is always prevented by line 12, so we restrict attention to the chance that FreshHTable$[h] = \perp$. Since line 19 is reached, at least one of (h, m_1) and (h, m_2) is in $\mathcal{F}$. In particular, FreshHTable$[h]$ was at some point initialized on line 11, but was at a later time set to $\perp$. Suppose that FreshHTable$[h]$ was initially set to (r^*, m^*); note that (r^*, m^*) would have been a chosen input to ADAPT. The entry FreshHTable$[h]$ only becomes set to $\perp$ if during some later query, $e^* = \mathsf{Emb}(r^*, m^*)$ was a random output of $\mathsf{BH.FP}$ on line 6, so that e^* was added to $\mathcal{E}$. If bad is set in G_1 then it must be the case that at least one "repeated" e^* was chosen over B's q queries. Since e^* only repeats if r^* does, it must be the case that a "repeated" r^* was chosen. This is bounded precisely by $\mathsf{R}^{-1}_{\mathsf{BH},\mathsf{Emb}}(q)$, which is the maximum probability, over all possible ADAPT inputs, that a randomly generated r' collided with a chosen-input r^*. This proves Eq. (19) and the theorem. ∎

D Forged certificate example

Suppose we have selected target hash $h =$ `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 $\mathtt{C}^* \neq \mathtt{C}$ with the same hash h; thus the signature on `C` can be reused.

A backdooring entity, following the specification of Section 6, does the following. Suppose they intend to use $\mathsf{BH} = \mathbf{HEB}[\mathsf{G}, \mathsf{S}, \mathsf{Emb}]$ where G is $\mathsf{SHA256}$, S is ECDSA over `secp256k1`, and

Certificate $\mathtt{C}^*$:

```
1  Data:
2     Version: 3 (0x2)
3     Serial Number: ...
4     Signature Algorithm: bdhWithRSAEncryption ; hk* = (SHA256, 04d0...a8a3)
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: bdhWithRSAEncryption ; hk* = (SHA256, 04d0...a8a3)
17 C.sigValue
```

Figure 26: A certificate forgery. The highlighted lines are the overhead in constructing a 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 backdooring adversary.

---

the embedding is as in Figure 14 of Section 6. We chose ECDSA because it is easily used in OpenSSL [40]; however a shorter signature scheme like BLS [10] may be easier to embed.

The forgery proceeds as follows. First, the backdoor is generated via $(hk^*, bd^*) \leftarrow\!\!\$\ \mathsf{BH.BKg}$, which in particular means generating (vk^*, sk^*) for ECDSA. (Recall that $hk^* = (gk^*, vk^*)$ and $bd^* = sk^*$ for **HEB**.) 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 6 discussed arbitrary data and in the Introduction, we discussed impersonation of `legit.com` by swapping out only the public key of $\mathtt{C.tbsCert}$. For this example, we address the former, and suppose that the big-brother adversary is simply aiming to forge $\mathtt{tbsCert}'$. Now they use $\mathsf{BH.FP}$ to find a preimage $\mathtt{tbsCert}^*$ of target hash h, where the constraint is that $\mathtt{tbsCert}^*$ is "close to" $\mathtt{tbsCert}'$. Concretely, $\mathtt{tbsCert}^*$ adds two additional fields. In the first, $\mathtt{tbsCert}^*.\mathtt{f}_h = h$. In the second, $\mathtt{tbsCert}^*.\mathtt{f}_\sigma = \sigma$, where $\sigma \leftarrow\!\!\$\ \mathsf{S.Sign}(sk^*, (h, \mathtt{tbsCert}'))$. For our chosen data and h, 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 $(\langle \mathsf{BH} \rangle, hk^*, \langle \mathsf{S_{ca}} \rangle)$. This is shown in Figure 26. Note that our threat model assumes big-brother has the power to cause $(\langle \mathsf{BH} \rangle, hk^*, \langle \mathsf{S_{ca}} \rangle)$ to be a valid algorithm specification.