A preliminary version of this paper, entitled "Hash functions from Sigma protocols and improvements to VSH," appears in *Advances in Cryptology - ASIACRYPT 2008*, Lecture Notes in Computer Science Vol. 5350 pp. 125–142, J. Pieprzyk ed., Springer-Verlag, 2008. This is the full version.

A Characterization of Chameleon Hash Functions and New, Efficient Designs

Mihir Bellare[1] Todor Ristov[2]

May 2011

Abstract

This paper shows that chameleon hash functions and Sigma protocols are equivalent. We provide a transform of any suitable Sigma protocol to a chameleon hash function, and also show that any chameleon hash function is the result of applying our transform to some suitable Sigma protocol. This enables us to unify previous designs of chameleon hash functions, seeing them all as emanating from a common paradigm, and also obtain new designs that are more efficient than previous ones. In particular, via a modified version of the Fiat-Shamir protocol, we obtain the fastest known chameleon hash function with a proof of security based on the *standard* factoring assumption. The increasing number of applications of chameleon hash functions, including on-line/off-line signing, chameleon signatures, designated-verifier signatures and conversion from weakly-secure to fully-secure signatures, make our work of contemporary interest.

[1]Department of Computer Science and Engineering, University of California at San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA. URL: `http://cseweb.ucsd.edu/~mihir/`. Supported in part by NSF grants CNS 0524765, CNS 6627779, CCF 0915675 and CNS 0904380.

[2]Qualcomm, 5775 Morehouse Drive, San Diego, CA 92121, USA. Work done while author was at UCSD, supported in part by above-mentioned grants of first author.

Contents

1 Introduction

The failure of popular hash functions MD5 and SHA-1 [54, 55] lends an impetus to the search for new ones. The contention of our paper is that there will be a "niche" market for as-fast-as-possible hash functions proven secure under standard assumptions. We provide a general paradigm that yields such functions.

The hash functions we get are chameleon [31] and we extend the treatment to get a characterization of chameleon hash functions, on the one hand unifying and clarifying previous constructs and on the other hand yielding new and more efficient ones. Let us now look at all this in more detail.

THE NEED FOR PROVEN-SECURE HASHING. Suppose an important document has been signed with a typical hash-then-sign scheme such as PKCS#1 [30]. If collisions are found in the underlying hash function the public key needs to be revoked and the signature can no longer be accepted. Yet there are instances in which we want a public key and signatures under it to survive for twenty or more years. This might be the case for a central and highly disseminated certificate or an important contract. Revocation of a widely disseminated public key is simply too costly and error-prone. In such a case, we want to be able to trust that collisions in our hash function will not be found even twenty years down the line.

Given the failure of MD5 and SHA-1, it would be understandable, from this twenty-year perspective, to feel uncertain about any hash function designed by "similar" methods. On the other hand, we may be very willing to pay a (reasonable!) computational price for security because documents or certificates of the ultra-importance we are considering may not need to be signed often. In this case, hash functions with *proven* security are interesting, and the faster they are the better. Our contribution is a general transform that yields a plurality of such hash functions, not only providing new ones but "explaining" or improving old ones.

FROM Σ TO HASH. We show how to construct a collision-resistant hash function from any (suitable) Σ-protocol. Recall that Σ-protocols are a class of popular 3-move identification schemes. Canonical examples are the Schnorr [46], Fiat-Shamir [21] and GQ [24] protocols, but there are many others as well [36, 39, 10, 25, 45, 38, 40]. Briefly, the protocols achieve a strong form of the usual honest-verifier zero-knowledge property, and our hash function is defined using the simulator. (We stress that the computation of the hash is deterministic even though the simulator is randomized.) The collision-resistance stems from strong special soundness [9], a well-studied property of Σ-protocols. The advantage of our approach is that there is a rich history in constructing proven-secure Σ-protocols and we can now leverage this to get collision-resistant hash functions. For future reference let us refer to a hash function derived from our approach as a Σ-hash function.

Damgard [19] and Cramer, Damgard and Mckenzie [16] have previously shown that it is possible to design commitment schemes based on Σ-protocols, but prior to our work it has not been observed that one can design collision-resistant hash functions from Σ-protocols. Note that secure commitment is not known to imply collision-resistant hashing and in fact is unlikely to do so because the former can be based on one-way functions [37] and the latter probably not [49]. Perhaps as a consequence, our construction requires slightly stronger properties from the Σ-protocols than do the constructions of [19, 16].

SPECIFIC DESIGNS. The Schnorr [46] and GQ [24] schemes are easily shown to meet our conditions, yielding collision resistant Σ-hash functions $\mathcal{H}$-$\mathcal{S}ch$ and $\mathcal{H}$-$\mathcal{G}\mathcal{Q}$ based, respectively, on discrete log and RSA. More interesting is the Fiat-Shamir protocol $\mathcal{FS}$ [21]. It doesn't satisfy strong special soundness

Pre	$\mathcal{H}$-$\mathcal{D}a$	$\mathcal{H}$-$\mathcal{ST}$	$\mathcal{H}$-$\mathcal{SFS}$
0	1	0.22	2
2048	1	0.33	4
16384	1	2	8

Figure 1: Performance of factoring-based hash functions. The modulus and output size are 1024 bits and the block size is 512 bits. "Pre" is the amount of pre-computation, in number of group elements stored. The table entry is the rate, defined as the average number of bits of data hashed per modular multiplication.

---

but we modify it to a protocol $\mathcal{SFS}$ (strong $\mathcal{FS}$) that we prove does under the factoring assumption, thereby obtaining a Σ-hash function $\mathcal{H}$-$\mathcal{SFS}$. From a modified version of the Micali-Shamir protocol [36] we obtain a Σ-hash function $\mathcal{H}$-$\mathcal{SMS}$ with security based on the SRPP (Square Roots of Prime Products) assumption of [36]. We also obtain a Σ-hash $\mathcal{H}$-$\mathcal{O}\mathcal{k}a$ from Okamoto's protocol [39] and a pairing-based Σ-hash $\mathcal{H}$-$\mathcal{HS}$ from an identification protocol of [5] derived from the identity-based signature scheme of Hess [25].

How fast? One question we consider interesting is, how fast can one hash while maintaining a proof of security under the *standard* factoring assumption? Figure 1 compares $\mathcal{H}$-$\mathcal{SFS}$ to the fastest known factoring-based functions and shows that the former emerges as the winner. (VSH, the Very Smooth Hash function of [14], is faster than all these, but is based on a non-standard assumption related to the difficulty of extracting modular square roots of products of small primes. We will discuss VSH, and our improvement to it, in a bit.) In Figure 1, $\mathcal{H}$-$\mathcal{D}a$ is the most efficient factoring-based instantiation known of Damgård's claw free permutation-based hash function [17, 23, 31]. $\mathcal{H}$-$\mathcal{ST}$ is the hash function of Shamir and Tauman [47]. The table entries are the rate, defined as the average number of bits of data hashed per modular multiplication in MD mode with a block size of 512 bits and a modulus and output size of 1024 bits. The figure shows that without pre-computation, $\mathcal{H}$-$\mathcal{SFS}$ is twice as fast as $\mathcal{H}$-$\mathcal{D}a$ and 9 times as fast as $\mathcal{H}$-$\mathcal{ST}$. But $\mathcal{H}$-$\mathcal{SFS}$ is amenable to pre-computation based speedup and $\mathcal{H}$-$\mathcal{D}a$ is not, so the gap in their rates increases swiftly with storage. $\mathcal{H}$-$\mathcal{ST}$ is also amenable to pre-computation based speedup but $\mathcal{H}$-$\mathcal{SFS}$ remains a factor 4 faster for any given amount of storage. We also remark that additionally $\mathcal{H}$-$\mathcal{SFS}$ is amenable to parallelization, unlike the other functions. We remark that $\mathcal{H}$-$\mathcal{SMS}$ is faster than $\mathcal{H}$-$\mathcal{SFS}$ but based on a stronger assumption. In Section 6 we recall $\mathcal{H}$-$\mathcal{D}a$ and $\mathcal{H}$-$\mathcal{ST}$ and justify the numbers in Figure 1. We also discuss implementation results.

Additional features. Σ-hash functions are keyed. While one can, of course, simply hardwire into the code a particular key to get an unkeyed function in the style of MD5 or SHA-1, it is advantageous, as explained in [7], to allow each user to choose their own key. The reason is that damage from a collision is now limited to the user whose key is involved, and the attacker must re-invest resources to attack another key. This slows down the rate of attacks and gives users time to get patches in place or revoke keys.

The reductions underlying the security proofs of Σ-hash functions are tight, so that the proven security guarantees hold with standard values of the security parameters.

Σ-hash functions are chameleon. Krawczyk and Rabin [31] introduced chameleon hash functions. The over 150 citations to date to their paper (as per Google Scholar) are an indication of the popularity

and utility of the primitive.

Krawczyk and Rabin [31] presented two example constructions of chameleon hash functions, and others were found by [47, 2, 3]. The analyses, however, are ad hoc. We, in contrast, show a general result, namely, that any Σ-hash is chameleon (cf. Theorem 5.1). As a consequence, we immediately obtain that $\mathcal{H}$-$\mathcal{GQ}$, $\mathcal{H}$-$\mathcal{SFS}$, $\mathcal{H}$-$\mathcal{SMS}$, $\mathcal{H}$-$\mathcal{Oka}$ and $\mathcal{H}$-$\mathcal{HS}$ are chameleon. In particular, in $\mathcal{H}$-$\mathcal{SFS}$, we obtain the fastest known chameleon hash function under the standard factoring assumption.

Shamir and Tauman used chameleon hash functions to build on-line/off-line signature schemes [47]. (The concept is due to [20].) This means that when one uses a Σ-hash one can completely eliminate the on-line cost of signing. (This cost is shifted entirely to the off-line phase.) Another application is chameleon signatures [31], which provides a recipient with a non-repudiable signature of a message without allowing it to prove to a third party that the signer signed this message. As explained in [31] this is an important tool for privacy-respecting authenticity in the signing of contracts and agreements. Chameleon hash functions are also used in designated-verifier signatures to achieve privacy [29, 50]. Finally, chameleon hashing can be used to transform a weakly-secure signature scheme into a fully-secure one. This is used in many places [31, 48, 11, 26] and a full statement and proof were provided by Hohenberger and Waters [27] whose design of RSA-based signatures made crucial use of this transform. By adding new and more efficient chameleon hash functions to the pool of existing ones we enable new and more efficient ways to implement all the different applications.

Reverse connection. As indicated above, we show that Σ-hash functions are chameleon. To complement this, we show that the converse is true as well, namely, all chameleon hash functions are Σ-hash functions (cf. Theorem 5.2). We prove this by associating to any chameleon hash function $\mathcal{H}$ a Σ-protocol $\mathcal{SP}$ such that applying our Σ2H (Σ-to-hash) transform to $\mathcal{SP}$ returns $\mathcal{H}$. We thereby have a characterization of chameleon hash functions as Σ-hash functions, which, as we discuss below, allows us to unify previous work.

We also obtain numerous new Σ-protocols, and thus identification protocols and, via [16, 19], commitment schemes, from existing chameleon hash functions such as $\mathcal{H}$-$\mathcal{Da}$ [17] and $\mathcal{H}$-$\mathcal{ST}$ [47]. However, we are not aware of any practical benefit of these constructs over known ones.

Unifying previous work. $\mathcal{H}$-$\mathcal{Sch}$ turns out to be exactly the classical hash function of Chaum, Van Heijst and Pfitzmann [13], which was shown to be chameleon by [31]. $\mathcal{H}$-$\mathcal{Oka}$ is an extension thereof [13]. $\mathcal{H}$-$\mathcal{GQ}$ is a special case of a chameleon hash function proposed by Ateniese and de Medeiros [2, 3]. (Our other hash functions $\mathcal{H}$-$\mathcal{SFS}$, $\mathcal{H}$-$\mathcal{SMS}$ and $\mathcal{H}$-$\mathcal{HS}$ are new.) The re-derivation of these hash functions as Σ-hashes sheds new light on the designs and shows how the Σ paradigm explains and unifies previous constructs.

Finally we make a connection between VSH [14] and $\mathcal{H}$-$\mathcal{SMS}$, the Σ-hash function emanating from the protocol of Micali and Shamir [36]. The latter is a more efficient version of the Fiat-Shamir protocol in which the public key, rather than consisting of random quadratic residues, consists of small primes. Interestingly $\mathcal{H}$-$\mathcal{SMS}$ turns out to be the VSH compression function [14] modulo some details. We suggest that this provides some intuition for the VSH design. It turns out that we can exploit this connection to get some improvements to VSH.

VSH*. In number-theoretic hashing there is (as elsewhere) a trade-off between speed and assumptions. We saw above that $\mathcal{H}$-$\mathcal{SFS}$ is the fastest known hash function under the standard factoring assumption. We now turn to non-standard factoring-related assumptions. Here the record-holder is VSH (Very Smooth Hash), a construct of Contini, Lenstra and Steinfeld [14] which has a proof of collision-resistance based on the VSSR assumption of the same paper [14]. We provide a modifica-

tion VSH* whose compression function, unlike that of the original, is collision resistant, leading to better performance for short messages. (Our implementations show that VSH* is up to 5 times faster than VSH on short messages. On long messages they have the same performance.) This is important because short messages are an important case in practice. (For example, most Internet packets are short.) VSH* remains provably collision-resistant under the same VSSR assumption as VSH. A different collision-resistant modification of the compression function of VSH is provided by [51].

We provide analogous improvements for the Fast-VSH variant of VSH provided by [14]. Again we can provide Fast-VSH* whose underlying compression function (unlike that of Fast-VSH) is proven collision-resistant, leading to speedups in hashing short messages. However, the speed gains are smaller than in the previous case.

Overall we believe that, even putting performance aside, having a collision resistant compression function underlying a hash function is a plus since it can be used directly and makes the hash function more misuse-resistant.

WHAT Σ-HASH FUNCTIONS AREN'T. Some recent work [15, 6, 1] suggests that general-purpose hash functions should have extra properties like pseudorandomness. Σ-hash functions are merely collision-resistant and chameleon; they do not offer these extra attributes. But as indicated above, Σ-hash functions are not intended to be general purpose. The envisaged applications are chameleon hashing and proven-secure, reasonable cost (purely) collision-resistant hashing.

2 Related work

Damgård [17] presents a construction of collision-resistant hash functions from claw-free permutation pairs [23]. As noted above, his factoring-based instantiation, based on [23] and also considered in [31, 47], is slower than our $\mathcal{H}$-$\mathcal{SFS}$.

Ishai, Kushilevitz and Ostrovsky [28] show how to transform homomorphic encryption (or commitment) schemes into collision-resistant hash functions. This is an interesting theoretical connection between the primitives. As far as we can tell, however, the approach is not yet practical. Specifically, their quadratic-residuosity (QR) based instantiation has a rate of 1/40 (that is, 40 modular multiplications per bit) with a 1024 bit modulus. (Their matrix needs 80 rows to get the 80-bit security corresponding to a 1024-bit modulus.) Hence their function is much slower than the constructs of Figure 1 in addition to being based on a stronger assumption (QR as opposed to factoring). Additionally it has a $80 \cdot 1024$ bit output so in a practical sense is not really hashing. Other instantiations of their construction that we know (El Gamal under DDH, Paillier [41] under DCRA) are also both slower than known ones and based on stronger assumptions.

Lyubashevsky, Micciancio, Peikert and Rosen [34] present a fast hash function SWIFFT with an asymptotic security proof based on assumptions about the hardness of lattice problems [42, 33], but the proof would not seem to yield guarantees for the parameter sizes proposed in [34]. In contrast, our reductions are tight and the proofs provide guarantees for standard values of the security parameters.

Bellare and Micciancio's construction [4] (whose goal was to achieve incrementality) uses random oracles, but these can be eliminated by using a small block size, such as one bit. In this case their MuHASH is provably collision-resistant based only on the discrete-log assumption, and runs at 0.33 bits per group operation in MD mode. In comparison, $\mathcal{H}$-$\mathcal{Sch}$ (also discrete log based) is faster, at 0.57 bits per group operation in MD mode.

Charles, Goren and Lauter [12] presented a hash function based on the assumed hardness of some problems related to elliptic curves. However, their construction was shown to not be collision-resistant

[52] and in fact to not even be pre-image resistant [43]. Tillich and Zémor [53] present a hash function based on the assumed hardness of some graph problems, whose security properties and efficiency were improved by Petit, Veyrat-Charvillon, and Quisquater [44]. The hash function of [44] is slower than Fast-VSH, and thereby slower than Fast-VSH*, according to the performance results reported in [14] and [44].

Heng and Kurosawa [32] define *reversible* Σ-protocols and show that these and trapdoor commitment schemes are equivalent. Reversibility, a property we do not assume, requires that the prover's randomness or internal state can be reconstructed from the public key and last two messages in the protocol given the secret key. The binding property of the commitment scheme is a weak form of collision resistance which rules out efficiently finding message-randomness pairs where the message parts are different but the corresponding commitments are the same. They do not claim or provide (chameleon) collision-resistant hash functions.

Steinfeld, Pieprzyk and Wang [51] suggest a collision-resistant modification of the VSH compression function based on restricting the domain of the second argument. This makes iterating the compression function somewhat less convenient but it can be done using the methods we discuss in Appendix A and would then appear to yield performance benefits similar to those we get via VSH*. They do not consider Fast-VSH.

Preliminary version. A preliminary version of this paper appeared as [8].

3 Definitions

Notation and conventions. We denote by $a_1\|\cdots\|a_n$ a string encoding of $a_1,\ldots,a_n$ from which the constituent objects are uniquely recoverable. We denote the empty string by ε. Unless otherwise indicated, an algorithm may be randomized. If A is a randomized algorithm then $y \leftarrow\!\!\$\, A(x_1,\ldots)$ denotes the operation of running A with fresh coins on inputs $x_1,\ldots$ and letting y denote the output. We denote by $[A(x_1,\ldots)]$ the set of all y that have positive probability of being output by A on input $x_1,\ldots$. If S is a (finite) set then $s \leftarrow\!\!\$\, S$ denotes the operation of picking s uniformly at random from S. If $X = x_1\|x_2\|\ldots\|x_n$, then $x_1\|x_2\|\ldots x_n \leftarrow X$ denotes the operation of parsing X into its constituents. Similarly, if $X = (x_1, x_2, \ldots, x_n)$ is an n-tuple, then $(x_1, x_2, \ldots, x_n) \leftarrow X$ denotes the operation of parsing X into its elements. We denote the security parameter by k, and by 1^k its unary encoding. Vectors are denoted in boldface, for example $\mathbf{u}$. If $\mathbf{u}$ is a vector then $|\mathbf{u}|$ is the number of its components and $\mathbf{u}[i]$ is its i-th component. "PT" stands for polynomial time.

Σ-protocols. A Σ-protocol is a three-move interactive protocol conducted by a prover and a verifier. Formally, it is a tuple $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathrm{CmSet}, \mathrm{ChSet}, \mathrm{RpSet})$, where K, P are PT algorithms and V is a deterministic boolean algorithm. The key-generation algorithm K takes input 1^k and returns a pair (pk, sk) consisting of a public and secret key for the prover. The latter is initialized with pk, sk while the verifier is initialized with pk. The parties interact as depicted in Figure 2. The prover begins by applying P to pk, sk to yield his first move $\mathrm{Y} \in \mathrm{CmSet}(pk)$, called the commitment, together with state information y, called the ephemeral secret key. The commitment is sent to the verifier, who responds with a challenge c drawn at random from $\mathrm{ChSet}(pk)$. The prover computes its response z by applying P to pk, sk, the challenge and the ephemeral secret key y. (This computation may use fresh coins although in the bulk of protocols it is deterministic.) Upon receiving c the verifier applies V to the public key and transcript $\mathrm{Y}\|\mathrm{c}\|\mathrm{z}$ of the conversation to decide whether to accept or reject. We require *completeness*, which means that an interaction between the honest prover and verifier is

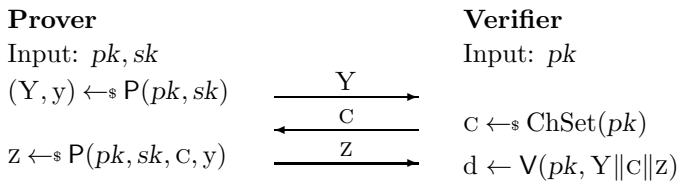

Figure 2: Σ-protocol. Keys pk and sk are produced using key-generation algorithm K.

---

always accepting. Formally, for all $k \in \mathbb{N}$ we have $\mathrm{d} = 1$ with probability 1 in the experiment

$$(pk, sk) \leftarrow\!\!\$\ \mathsf{K}(1^k);\ (\mathrm{Y}, \mathrm{y}) \leftarrow\!\!\$\ \mathsf{P}(pk, sk);\ \mathrm{c} \leftarrow\!\!\$\ \mathrm{ChSet}(pk);$$

$$\mathrm{z} \leftarrow\!\!\$\ \mathsf{P}(pk, sk, \mathrm{c}, \mathrm{y});\ \mathrm{d} \leftarrow \mathsf{V}(pk, \mathrm{Y}\|\mathrm{c}\|\mathrm{z}).$$

The verifier given $pk, \mathrm{Y}\|\mathrm{c}\|\mathrm{z}$ should always check that $\mathrm{Y} \in \mathrm{CmSet}(\mathrm{Y})$ and $\mathrm{c} \in \mathrm{ChSet}(pk)$ and $\mathrm{z} \in \mathrm{RpSet}(pk)$ and reject otherwise. We implicitly assume this is done throughout.

Security notions. We provide formal definitions of strong special soundness (sss) and strong honest verifier zero-knowledge (StHVZK). Strong special soundness of Σ-protocol $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathrm{CmSet}, \mathrm{ChSet}, \mathrm{RpSet})$ [9] asks that it be computationally infeasible, given only the public key, to produce a pair of accepting transcripts that are commitment-agreeing but challenge-response-disagreeing. Formally an sss-adversary, on input pk, returns a tuple $(\mathrm{Y}, \mathrm{c}_1, \mathrm{z}_1, \mathrm{c}_2, \mathrm{z}_2)$ such that $\mathrm{Y} \in \mathrm{CmSet}(pk); \mathrm{c}_1, \mathrm{c}_2 \in \mathrm{ChSet}(pk)$; $\mathrm{z}_1, \mathrm{z}_2 \in \mathrm{RpSet}(pk)$ and $(\mathrm{c}_1, \mathrm{z}_1) \neq (\mathrm{c}_2, \mathrm{z}_2)$. The advantage $\mathbf{Adv}^{\mathrm{sss}}_{\mathcal{SP},A}(k)$ of such an adversary is defined for all $k \in \mathbb{N}$ as the probability that $\mathsf{V}(pk, \mathrm{Y}\|\mathrm{c}_1\|\mathrm{z}_1) = 1$ and $\mathsf{V}(pk, \mathrm{Y}\|\mathrm{c}_2\|\mathrm{z}_2) = 1$ in the experiment where $\mathsf{K}(1^k)$ is first executed to get (pk, sk) and then $A(pk)$ is executed to get $(\mathrm{Y}, \mathrm{c}_1, \mathrm{z}_1, \mathrm{c}_2, \mathrm{z}_2)$. We say that $\mathcal{SP}$ has strong special soundness if $\mathbf{Adv}^{\mathrm{sss}}_{\mathcal{SP},A}(\cdot)$ is negligible for all PT sss-adversaries A. To define StHVZK, let $\mathcal{TR}_{\mathcal{SP}}$ be the algorithm that on input (pk, sk) executes P and V as per Figure 2 and returns the transcript $\mathrm{Y}\|\mathrm{c}\|\mathrm{z}$. Recall that a PT algorithm Sim is a HVZK simulator for $\mathcal{SP}$ if the outputs of the processes

$$(pk, sk) \leftarrow\!\!\$\ \mathsf{K}(1^k);\ \ \mathbf{Return}\ (pk, \mathsf{Sim}(pk))$$

and

$$(pk, sk) \leftarrow\!\!\$\ \mathsf{K}(1^k);\ \ \mathbf{Return}\ (pk, \mathcal{TR}_{\mathcal{SP}}(pk, sk))$$

are identically distributed. (We require perfect, not computational, ZK. This simplifies applications and there is no particular loss from assuming it since it is provided by the candidate protocols.) We say that a PT algorithm StSim is a strong HVZK (StHVZK) simulator for $\mathcal{SP}$ if StSim is deterministic and the algorithm Sim defined on input pk by

$$\mathrm{c} \leftarrow\!\!\$\ \mathrm{ChSet}(pk);\ \mathrm{z} \leftarrow\!\!\$\ \mathrm{RpSet}(pk);\ \mathrm{Y} \leftarrow \mathsf{StSim}(pk, \mathrm{c}, \mathrm{z});\ \mathbf{Return}\ \mathrm{Y}||\mathrm{c}||\mathrm{z}$$

is a HVZK simulator for $\mathcal{SP}$. We say that $\mathcal{SP}$ is StHVZK if it has a PT StHVZK simulator and also the commitment Y generated via $(\mathrm{Y}, \mathrm{y}) \leftarrow\!\!\$\ \mathsf{P}(pk, sk)$ is uniformly distributed over $\mathrm{CmSet}(pk)$ for all $(pk, sk) \in [\mathsf{K}(1^k)]$. We denote by $\boldsymbol{\Sigma}(\mathrm{sss})$ the set of all Σ-protocols that satisfy strong special soundness and by $\boldsymbol{\Sigma}(\mathrm{StHVZK})$ the set of all Σ-protocols that are strong HVZK.

Discussion. While the basic format of Σ-protocols as 3-move protocols of the type above is agreed upon, when it comes to security properties, there are different choices and variations in the literature. Our formalization of strong special soundness is from [9]. Strong HVZK seems to be new but the canonical protocols in this area have this property.

COLLISION-RESISTANT HASH FUNCTIONS. A family of n-input hash functions (where $n \geq 1$ is a constant) is a tuple $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{D}_1, \ldots, \mathrm{D}_\mathrm{n}, \mathrm{R})$. The key-generation algorithm KG takes input 1^k and returns a key K describing a particular function $\mathsf{H}_K : \mathrm{D}_1(K) \times \ldots \mathrm{D}_\mathrm{n}(K) \to \mathrm{R}(K)$. As this indicates, $\mathrm{D}_1, \ldots, \mathrm{D}_\mathrm{n}, \mathrm{R}$ are functions that given K return sets. A cr-adversary, on input K returns distinct tuples $(x_1, \ldots, x_n), (y_1, \ldots, y_n)$ such that $x_i, y_i \in \mathrm{D}_i(K)$ for all $1 \leq i \leq n$. The advantage $\mathbf{Adv}^{\mathrm{cr}}_{\mathcal{H},B}(k)$ of such an adversary B is defined for all $k \in \mathbb{N}$ as the probability that $\mathsf{H}(K, x_1, \ldots, x_n) = \mathsf{H}(K, y_1, \ldots, y_n)$ in the experiment where $\mathsf{KG}(1^k)$ is first executed to get K and then $B(K)$ is executed to get $((x_1, \ldots, x_n), (y_1, \ldots, y_n))$. We say that $\mathcal{H}$ is collision resistant if the cr-advantage of any PT adversary B is negligible.

4 Σ-hash theory

This section covers the theory of Σ-hash functions. We present and justify the $\Sigma\mathsf{2H}$ transform that turns a Σ-protocol $\mathcal{SP} \in \Sigma(\mathrm{sss}) \cap \Sigma(\mathrm{StHVZK})$ into a collision-resistant hash function $\mathcal{H}$-$\mathcal{SP}$. Then we find Σ-protocols which we can prove have the required properties and derive specific Σ-hash functions. In Section 5 we relate Σ and chameleon hash functions. In Section 6 we discuss the practical and performance aspects of our Σ-hash functions.

4.1 The transform

We show how to build a collision-resistant hash function from any Σ-protocol $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathrm{CmSet}, \mathrm{ChSet}, \mathrm{RpSet}) \in \Sigma(\mathrm{sss}) \cap \Sigma(\mathrm{StHVZK})$ that satisfies strong special soundness and strong HVZK. Let StSim be a strong HVZK simulator for $\mathcal{SP}$. Let $\mathsf{K}^{(1)}$ be the algorithm that on input 1^k lets $(pk, sk) \leftarrow\!\!\$\ \mathsf{K}(1^k)$ and returns pk. We define the 2-input family of hash functions $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{ChSet}, \mathrm{RpSet}, \mathrm{CmSet})$ by $\mathsf{KG} = \mathsf{K}^{(1)}$ and $\mathsf{H}_{pk}(\mathrm{C}, \mathrm{Z}) = \mathsf{StSim}(pk, \mathrm{C}, \mathrm{Z})$. In other words, the key is the prover's public key. (The secret key is discarded.) The inputs to the hash function are regarded as the challenge and response in the Σ-protocol. The output is the corresponding commitment. The existence of a StHVZK simulator is exploited to *deterministically* compute this output. We refer to a family of functions defined in this way as a Σ-hash. We write $\mathcal{H} = \Sigma\mathsf{2H}(\mathcal{SP})$ to indicate that $\mathcal{H}$ has been derived as above from Σ-protocol $\mathcal{SP}$. The following theorem says that a Σ-hash family is collision-resistant.

Theorem 4.1 Let $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathrm{CmSet}, \mathrm{ChSet}, \mathrm{RpSet}) \in \Sigma(\mathrm{sss}) \cap \Sigma(\mathrm{StHVZK})$ be a Σ-protocol. Let $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{ChSet}, \mathrm{RpSet}, \mathrm{CmSet}) = \Sigma\mathsf{2H}(\mathcal{SP})$ be the family of hash functions associated to $\mathcal{SP}$ as above. For every cr adversary B against $\mathcal{H}$ there exists an sss-adversary A against $\mathcal{SP}$ such that for all k we have $\mathbf{Adv}^{\mathrm{cr}}_{H,B}(k) \leq \mathbf{Adv}^{\text{sss-na}}_{\mathcal{SP},A}(k)$, and the running time of B is that of A. ▌

The proof of this theorem, given below, is simple, but we note some subtleties, which is the way it relies on the (strong) HVZK and completeness of the Σ-protocol in addition to the strong special soundness.

Proof of Theorem 4.1: We define adversary A as follows.

Adversary $A(pk)$
$((\mathrm{C}_1, \mathrm{Z}_1), (\mathrm{C}_2, \mathrm{Z}_2)) \leftarrow\!\!\$\ B(pk)$; $\mathrm{Y} \leftarrow \mathsf{H}_{pk}(\mathrm{C}_1, \mathrm{Z}_1)$
Return $(\mathrm{Y}, \mathrm{C}_1, \mathrm{Z}_1, \mathrm{C}_2, \mathrm{Z}_2)$

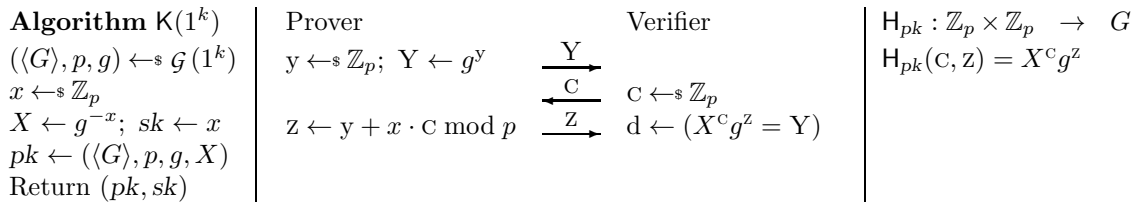

Figure 3: $\mathcal{Sch}$ Σ-protocol and the derived Σ-hash family, where $\mathcal{G}$ is a prime-order group generator.

By definition of a cr-adversary we know that $(\mathrm{c}_1, \mathrm{z}_1) \neq (\mathrm{c}_2, \mathrm{z}_2)$. Hence A satisfies the definition of an sss adversary. Let $\mathrm{Y}_i = \mathsf{H}_{pk}(\mathrm{c}_i, \mathrm{z}_i)$ for $i = 1, 2$. The definition of a cr-adversary also implies that $\mathrm{c}_i \in \mathrm{ChSet}(pk)$ and $\mathrm{z}_i \in \mathrm{RpSet}(pk)$ for $i = 1, 2$. Strong HVZK now implies that the transcripts $\mathrm{Y}_i \| \mathrm{c}_i \| \mathrm{z}_i$ have positive probability of being produced in the protocol, meaning of being output by $\mathcal{T}r_{\mathrm{SP}}(pk, sk)$. The completeness of the protocol now implies that $\mathsf{V}(pk, \mathrm{Y}_1 \| \mathrm{c}_1 \| \mathrm{z}_1) = 1$ and $\mathsf{V}(pk, \mathrm{Y}_2 \| \mathrm{c}_2 \| \mathrm{z}_2) = 1$. Finally, if B succeeds then $\mathrm{Y}_1 = \mathrm{Y}_2$ so A also succeeds. ▮

To construct Σ-hash functions we now seek Σ-protocols which we can show are in $\mathbf{\Sigma}(\mathrm{sss}) \cap \mathbf{\Sigma}(\mathrm{StHVZK})$.

4.2 Overview of constructions

We begin, as illustrative examples, with the Schnorr [46] and GQ [24] Σ-protocols, which we can easily show to have the desired properties. The hash functions obtained are known [13, 2, 3] and their re-derivation as Σ-hashes sheds new light on their design and also shows how the Σ-hash paradigm unifies and explains existing work. More interesting is the Fiat-Shamir [21] Σ-protocol. It doesn't satisfy strong special soundness, but we modify it to a Σ-protocol $\mathcal{SFS}$ that we prove is in $\mathbf{\Sigma}(\mathrm{sss}) \cap \mathbf{\Sigma}(\mathrm{StHVZK})$ under the standard factoring assumption. With non-standard factoring-related assumptions (that it is hard to extract modular square roots of products of small primes) we get a faster Σ-hash $\mathcal{H}$-$\mathcal{SMS}$ from a modification of the Micali-Shamir Σ-protocol [36]. We also get another discrete-log based Σ-hash from Okamoto's protocol [39] and a pairing based one from the $\mathcal{HS}$ protocol [25, 5]. Let us now detail all this.

4.3 $\mathcal{Sch}$

We fix a prime-order group generator, by which we mean a PT algorithm $\mathcal{G}$ that on input 1^k returns the description $\langle G\rangle$ of a group G of prime order $p \in \{2^{k-1}, \ldots, 2^k - 1\}$ together with p and a generator g of G. The key-generation process and protocol underlying the $\mathcal{Sch}$ Σ-protocol of [46] are then as shown in Figure 3. The algorithm that on input $pk = (\langle G\rangle, p, g, X)$ picks $\mathrm{c}, \mathrm{z} \leftarrow\!\!\$\ \mathbb{Z}_p$ and returns $X^c g^z \| \mathrm{c} \| \mathrm{z}$ is a HVZK simulator for $\mathcal{Sch}$, so $\mathcal{Sch} \in \mathbf{\Sigma}(\mathrm{StHVZK})$ and the derived Σ-hash $\mathcal{H}$-$\mathcal{Sch}$ is as shown in Figure 3. The key observation for strong special soundness is that if $X^{\mathrm{c}_1} g^{\mathrm{z}_1} = X^{\mathrm{c}_2} g^{\mathrm{z}_2}$ and $(\mathrm{c}_1, \mathrm{z}_1) \neq (\mathrm{c}_2, \mathrm{z}_2)$ then it must be that $\mathrm{c}_1 \neq \mathrm{c}_2$. This leads us to associate to sss-adversary A the discrete log finder D that on input $\langle G\rangle, p, g, X$ runs A on the same input to get $(\mathrm{Y}, \mathrm{c}_1, \mathrm{z}_1, \mathrm{c}_2, \mathrm{z}_2)$ and returns $(\mathrm{z}_2 - \mathrm{z}_1)(\mathrm{c}_1 - \mathrm{c}_2)^{-1} \bmod p$. Then for all k we have $\mathbf{Adv}^{\mathrm{sss}}_{\mathcal{Sch},A}(k) \leq \mathbf{Adv}^{\mathrm{dl}}_{\mathcal{G},D}(k)$, where the latter is defined as the probability that $x' = x$ in the experiment where we let $(\langle G\rangle, p, g) \leftarrow\!\!\$\ \mathcal{G}(1^k)$ and $x \leftarrow\!\!\$\ \mathbb{Z}_p$ and then let $x' \leftarrow\!\!\$\ D(\langle G\rangle, p, g, g^x)$. This shows that $\mathcal{Sch}$ has strong special soundness as long as the discrete log problem is hard relative to $\mathcal{G}$. By Theorem 4.1 $\mathcal{H}$-$\mathcal{Sch}$ is collision-resistant under the same assumption.

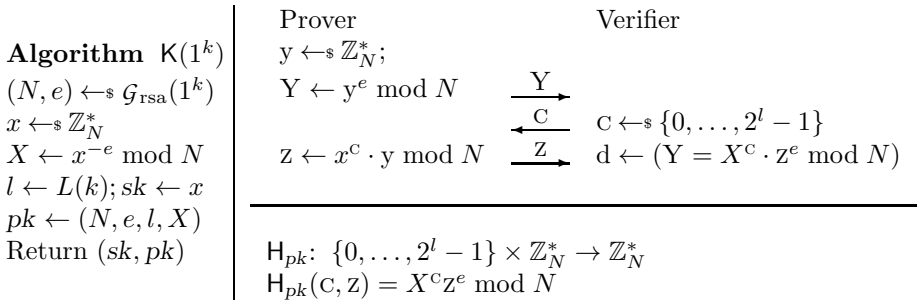

Figure 4: $\mathcal{GQ}$ Σ-protocol and the derived Σ-hash family, where $\mathcal{G}_{\rm rsa}$ is a prime exponent RSA generator with associated challenge length L.

4.4 $\mathcal{GQ}$

We fix a prime-exponent RSA generator with associated challenge length $L(\cdot)$, by which we mean a PT algorithm $\mathcal{G}_{\rm rsa}$ that on input 1^k returns an RSA modulus $N \in \{2^{k-1},\ldots,2^k-1\}$ and an RSA encryption exponent $e > 2^{L(k)}$ that is a prime. The key-generation process and protocol underlying Σ-protocol $\mathcal{GQ}$ of [24] are then as shown in Figure 4. The algorithm that on input $pk = (N,e,l,X)$ picks $\mathrm{C} \leftarrow\!\!\$\ \{0,1\}^l$; $\mathrm{z} \leftarrow\!\!\$\ Z_N^*$ and returns $\mathrm{Y}\|\mathrm{C}\|\mathrm{z}$, where $\mathrm{Y} = X^{\mathrm{C}}\mathrm{z}^e \bmod N$, is a HVZK simulator for $\mathcal{GQ}$, so $\mathcal{GQ} \in \Sigma(\mathrm{StHVZK})$ and the derived Σ-hash $\mathcal{H}$-$\mathcal{GQ}$ is as shown in Figure 4. Again observe that if $X^{\mathrm{C}_1}\mathrm{z}_1^e = X^{\mathrm{C}_2}\mathrm{z}_2^e$ and $(\mathrm{C}_1,\mathrm{z}_1) \neq (\mathrm{C}_2,\mathrm{z}_2)$ then $\mathrm{C}_1 \neq \mathrm{C}_2$. To adversary A attacking the strong special soundness, this leads us to associate the inverter I that on input N,e,X runs A on input N,e,l,X where $l = L(\lfloor \log_2(N) \rfloor + 1)$ to get $(\mathrm{Y},\mathrm{C}_1,\mathrm{z}_1,\mathrm{C}_2,\mathrm{z}_2)$ and returns $(\mathrm{z}_2\mathrm{z}_1^{-1})^b X^a \bmod N$ where a,b satisfy $ae + b(\mathrm{C}_1 - \mathrm{C}_2) = 1$ and are found via the extended gcd algorithm. (This is where we use the fact that e is prime.) Then for all k we have $\mathbf{Adv}^{\rm sss}_{\mathcal{GQ},A}(k) \le \mathbf{Adv}^{\rm rsa}_{\mathcal{G}_{\rm rsa},I}(k)$, where the latter is defined as the probability that $x' = x$ in the experiment where we let $(N,e) \leftarrow\!\!\$\ \mathcal{G}_{\rm rsa}(1^k)$ and $x \leftarrow\!\!\$\ \mathbb{Z}_N^*$ and then let $x' \leftarrow\!\!\$\ I(N,e,x^e \bmod N)$. This shows that $\mathcal{GQ}$ has strong special soundness if RSA is one-way relative to $\mathcal{G}_{\rm rsa}$. By Theorem 4.1, $\mathcal{H}$-$\mathcal{GQ}$ is collision-resistant under the same assumption.

4.5 $\mathcal{FS}$ and $\mathcal{SFS}$

We fix a modulus generator, namely a PT algorithm $\mathcal{G}_{\rm mod}$ that on input 1^k returns a modulus $N \in \{2^{k-1},\ldots,2^k-1\}$ and distinct primes p,q such that $N = pq$. We also fix a challenge length $L(\cdot)$. If C is a l-bit string and $\mathbf{u} \in (Z_N^*)^l$ then we let $\mathbf{u}^{\mathrm{C}} = \prod \mathbf{u}[i]^{\mathrm{C}[i]}$ where the product is over $1 \le i \le l$ and $\mathrm{C}[i]$ denotes the i-th bit of C. The key-generation algorithm and protocol underlying the $\mathcal{FS}$ Σ-protocol are then as shown in Figure 5. However this protocol does not satisfy strong special soundness because if $\mathrm{Y}\|\mathrm{C}\|\mathrm{z}$ is an accepting transcript relative to $pk = (N,l,\mathbf{u})$ then so is $\mathrm{Y}\|\mathrm{C}\|\mathrm{z}'$ where $\mathrm{z}' = N - \mathrm{z}$. We now show how to modify $\mathcal{FS}$ so that it has strong special soundness. First, some notation. For $w \in \mathbb{Z}_N$ we let $[w]_N$ equal w if $w \le N/2$ and $N - w$ otherwise. Let $\mathbb{Z}_N^+ = \mathbb{Z}_N^* \cap \{1,\ldots,N/2\}$. The modified protocol $\mathcal{SFS}$ (Strong $\mathcal{FS}$) is shown in Figure 5. Here $\mathrm{CmSet}((N,l,\mathbf{u}))$ is the set QR_N of quadratic residues in $\mathbb{Z}_N^*$ and $\mathrm{ChSet}((N,l,\mathbf{u}))$ is $\{0,1\}^l$, just as in $\mathcal{FS}$, but $\mathrm{RpSet}((N,l,\mathbf{u}))$ is now equal to $\mathbb{Z}_N^+$ rather than $\mathbb{Z}_N^*$ as before. For any adversary F we define $\mathbf{Adv}^{\rm fac}_{\mathcal{G}_{\rm mod},F}(k)$ as the probability that $r \in \{p,q\}$ in the experiment where we let $(N,p,q) \leftarrow\!\!\$\ \mathcal{G}_{\rm mod}(1^k)$ and $r \leftarrow\!\!\$\ F(N)$. The following shows that $\mathcal{SFS}$ has strong special soundness under the *standard* hardness of factoring assumption.

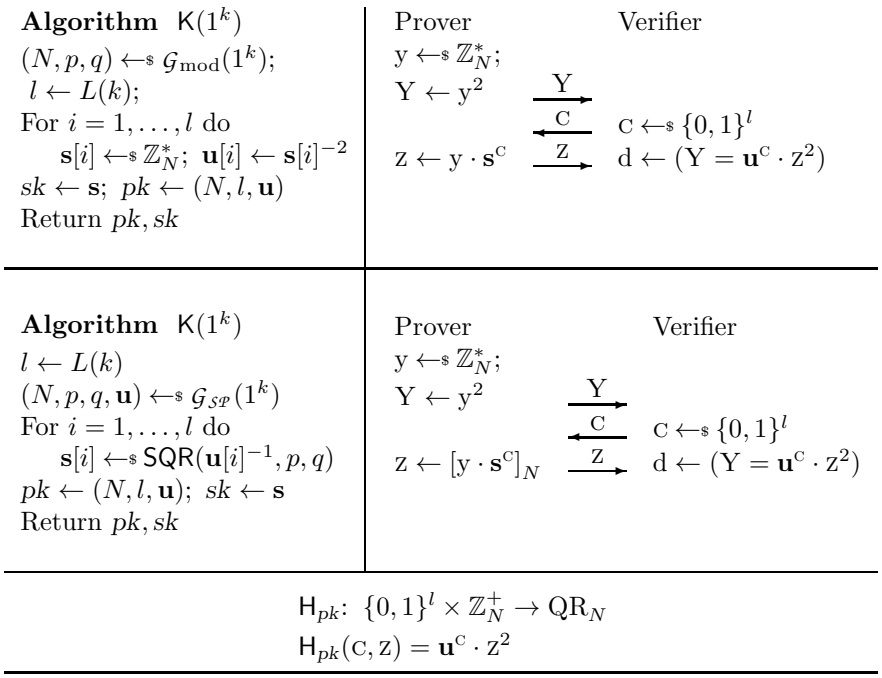

Figure 5: $\mathcal{FS}$, $\mathcal{SFS}$, $\mathcal{MS}$ and $\mathcal{SMS}$ protocols and the Σ-hash derived from $\mathcal{SFS}$, $\mathcal{SMS}$. The upper left key-generation algorithm is that of $\mathcal{FS}$ and $\mathcal{SFS}$, while the lower left one is that of $\mathcal{MS}$ and $\mathcal{SMS}$. The upper protocol is that of $\mathcal{FS}$ and $\mathcal{MS}$ while the lower protocol is that of $\mathcal{SFS}$ and $\mathcal{SMS}$. Here $\mathcal{G}_{\rm mod}$ is a modulus generator and $\mathcal{G}_{\rm SP}$ is a small prime modulus generator. The computations are in $\mathbb{Z}_N^*$, meaning modulo N.

Proposition 4.2 Let $\mathcal{G}_{\rm mod}$ be a modulus generator and $L(\cdot)$ a challenge length. Let $\mathcal{SFS}$ be the associated Σ-protocol as per Figure 5. If A is a sss-adversary against $\mathcal{SFS}$ then there is a factoring algorithm F against $\mathcal{G}_{\rm mod}$ such that for all k we have $\mathbf{Adv}^{\rm sss}_{\mathcal{SFS},A}(k) \le 2 \cdot \mathbf{Adv}^{\rm fac}_{\mathcal{G}_{\rm mod},F}(k)$. The running time of F is that of A plus the time for at most $L(\cdot)$ multiplications, one inversion modulo N, and the time for one execution of the gcd algorithm.

Proof: The factoring algorithm F is shown in Figure 6. For the analysis, consider two cases. The first is that $\mathrm{c}_1 \ne \mathrm{c}_2$ and the second is that $\mathrm{c}_1 = \mathrm{c}_2$ and $\mathrm{z}_1 \ne \mathrm{z}_2$. In the first case, a simple computation shows that $r_1^2 \equiv r_2^2 \pmod N$. On the other hand, $s[g]$ is chosen at random in $\mathbb{Z}_N^*$ and the only information A gets about it is $\mathbf{u}[g] = s[g]^{-2} \bmod N$ so $\mathbf{s}[g] \notin \{r_1, N - r_1\}$ with probability 1/2. So in this case F succeeds with probability 1/2 times the probability that A succeeds. In the case $\mathrm{c}_1 = \mathrm{c}_2$ but $\mathrm{z}_1 \ne \mathrm{z}_2$ we have, modulo N,

$$\mathrm{Y} \equiv \mathrm{z}_1^2 \cdot \textstyle\prod_{i=1}^{l} \mathbf{s}[i]^{\mathrm{c}_1[i]} \equiv \mathrm{z}_2^2 \cdot \prod_{i=1}^{l} \mathbf{s}[i]^{\mathrm{c}_2[i]}$$

and $\mathrm{c}_1 = \mathrm{c}_2$ implies $\mathrm{z}_1^2 \equiv \mathrm{z}_2^2$. But $\mathrm{z}_1 \ne \mathrm{z}_2$ and $\mathrm{z}_1, \mathrm{z}_2 \in \mathbb{Z}_N^+$ so it must be that $\mathrm{z}_1 \notin \{\mathrm{z}_2, N - \mathrm{z}_2\}$. So F succeeds with the same probability as A in this case. ∎

Now, the algorithm that on input $pk = (N,l,\mathbf{u})$ lets $\mathrm{C} \leftarrow\!\!\$\ \{0,1\}^l$; $\mathrm{Z} \leftarrow\!\!\$\ Z_N^+$, and $\mathrm{Y} \leftarrow \mathbf{u}^{\mathrm{C}} \cdot \mathrm{Z}^2 \bmod N$ and returns $\mathrm{Y}\|\mathrm{C}\|\mathrm{Z}$ is a HVZK simulator for $\mathcal{SFS}$. Accordingly $\mathcal{SFS} \in \Sigma(\mathrm{StHVZK})$ and we derive from $\mathcal{SFS}$ the Σ-hash family $\mathcal{H}$-$\mathcal{SFS}$ shown in Figure 5. Proposition 4.2 and Theorem 4.1 imply that $\mathcal{H}$-$\mathcal{SFS}$ is collision resistant under the standard factoring assumption.

Algorithm $F(N)$
For $i = 1, \ldots, n$ do
  $\mathbf{s}[i] \leftarrow\!\!\$\ \mathbb{Z}_N^*$; $\mathbf{u}[i] \leftarrow \mathbf{s}[i]^{-2} \bmod N$
$l \leftarrow L(k)$; $pk \leftarrow (N, l, \mathbf{u})$; $(\mathrm{Y}, \mathrm{C}_1, \mathrm{Z}_1, \mathrm{C}_2, \mathrm{Z}_2) \leftarrow\!\!\$\ A(pk)$
If $\mathrm{C}_1 \neq \mathrm{C}_2$ then
  $g \leftarrow\!\!\$\ \{1, \ldots, l : \mathrm{C}_1[i] \neq \mathrm{C}_2[i]\}$
  $r_1 \leftarrow \left(\frac{\mathrm{Z}_1}{\mathrm{Z}_2} \cdot \prod_{i \neq g} \mathbf{s}[i]^{\mathrm{C}_2[i] - \mathrm{C}_1[i]}\right)^{\mathrm{C}_1[g] - \mathrm{C}_2[g]}$
  $r_2 \leftarrow \mathbf{s}[g]$
Else // $\mathrm{C}_1 = \mathrm{C}_2$ and $\mathrm{Z}_1 \neq \mathrm{Z}_2$
  $r_1 \leftarrow \mathrm{Z}_1$; $r_2 \leftarrow \mathrm{Z}_2$
$r \leftarrow \gcd(N, r_1 - r_2)$
Return r

Adversary $B(N, \mathbf{u})$
$pk \leftarrow (N, \mathbf{u})$; $(\mathrm{Y}, \mathrm{C}_1, \mathrm{Z}_1, \mathrm{C}_2, \mathrm{Z}_2) \leftarrow\!\!\$\ A(pk)$
If $\mathrm{C}_1 \neq \mathrm{C}_2$
  $T \leftarrow \{j : \mathrm{C}_1[j] = 1 \wedge \mathrm{C}_2[j] = 0\}$
  $R \leftarrow \{j : \mathrm{C}_1[j] = 1 \wedge \mathrm{C}_2[j] = 1\}$
  $M \leftarrow \{j : \mathrm{C}_1[j] = 0 \wedge \mathrm{C}_2[j] = 1\}$; $S \leftarrow M \cup T$
  $x \leftarrow \frac{\mathrm{Z}_1}{\mathrm{Z}_2} \prod_{j \in T} \mathbf{u}[j]$
Else // $\mathrm{C}_1 = \mathrm{C}_2$ and $\mathrm{Z}_1 \neq \mathrm{Z}_2$
  $p \leftarrow \gcd(N, \mathrm{Z}_1 - \mathrm{Z}_2)$ // p is a factor of N
  $x \leftarrow \mathsf{SQR}(\mathbf{u}[1], p, N/p) \bmod N$; $S \leftarrow \{1\}$
Return (x, S)

Figure 6: Factoring algorithm F for proof of Proposition 4.2 and spr-adversary B for proof of Proposition 4.3.

4.6 $\mathcal{MS}$ and $\mathcal{SMS}$

The Micali-Shamir protocol [36] is a variant of $\mathcal{FS}$ in which verification time is reduced by choosing the coordinates of $\mathbf{u}$ to be small primes. As with $\mathcal{FS}$ it does not satisfy sss, but we can modify it to do so and thereby obtain a collision-resistant hash function $\mathcal{H}$-$\mathcal{SMS}$ that is faster than $\mathcal{H}$-$\mathcal{SFS}$ at the cost of a stronger assumption for security. To detail all this, let $\mathcal{G}_{\mathcal{SP}}$ be a small prime modulus generator with challenge length $L(\cdot)$, by which we mean a PT algorithm that on input 1^k returns a modulus $N \in \{2^{k-1}, \ldots, 2^k - 1\}$, distinct primes p, q such that $N = pq$, and an $L(k)$-vector $\mathbf{u}$ each of whose coordinates is a prime in $\mathrm{QR}(N) = \{x^2 \bmod N : x \in Z_N^*\}$. For efficiency we would choose these primes to be as small as possible. (For example $\mathbf{u}[i]$ is the i-th prime in $\mathrm{QR}(N)$.) An spr-adversary B against $\mathcal{G}_{\mathrm{SP}}, L$ takes input N and $\mathbf{u} \in (\mathbb{Z}_N^*)^{L(k)}$ and returns (x, S) where $x \in \mathbb{Z}_N^*$ and S is a non-empty subset of $\{0,1\}^l$. Its spr-advantage is defined for all k by

$$\mathbf{Adv}^{\mathrm{spr}}_{\mathcal{G}_{\mathrm{SP}}, L, B}(k) = \Pr\left[\, x^2 \equiv \textstyle\prod_{i \in S} \mathbf{u}[i] \pmod N : (N, p, q, \mathbf{u}) \leftarrow\!\!\$\ \mathcal{G}_{\mathrm{SP}}(1^k)\ ; (x, S) \leftarrow\!\!\$\ B(N, \mathbf{u}) \,\right].$$

The SRPP (Square Root of Prime Products) assumption [36] says that the spr-advantage of any PT B is negligible. Now, Figure 5 shows our modified version $\mathcal{SMS}$ of the Micali-Shamir protocol. It is in Σ(StHVZK) for the same reason as $\mathcal{SFS}$ and hence the derived hash function is again as shown, where $\mathsf{SQR}(\cdot, p, q)$ takes input $w \in \mathrm{QR}(N)$ and returns at random one of the four square roots of w modulo $N = pq$, computed using the primes p, q. Strong special soundness of $\mathcal{SMS}$ is proven in the following under the SRPP assumption.

Proposition 4.3 Let $\mathcal{G}_{\mathcal{SP}}$ be a small prime modulus generator with associated challenge length L. Let $\mathcal{SMS}$ be the associated Σ-protocol as per Figure 5. If A is a sss-adversary against $\mathcal{SMS}$ then there is a spr-adversary B such that for all k we have $\mathbf{Adv}^{\mathrm{sss}}_{\mathcal{G}_{\mathcal{SP}}, L, A}(k) \leq \mathbf{Adv}^{\mathrm{spr}}_{\mathcal{G}_{\mathcal{SP}}, L, B}(k)$. The running time of B is that of A plus time $t = \max\{t_1, t_2\}$, where t_1 is the time it takes to execute one inversion modulo N and $L + 1$ multiplications modulo N and t_2 is the time it takes to execute the gcd algorithm and the SQR algorithm.

Proof: Let's explain why the adversary B shown in Figure 6 works. If A succeeds then

$$\mathrm{z}_1{}^2 \prod_{j=1}^{n} \mathbf{u}[j]^{\mathrm{c}_1[j]} \equiv \mathrm{z}_2{}^2 \prod_{j=1}^{n} \mathbf{u}[j]^{\mathrm{c}_2[j]} \pmod{N} \quad (1)$$

Now, when $\mathrm{c}_1 \neq \mathrm{c}_2$, multiplying both sides of this equation by $\prod_{j\in T}\mathbf{u}[j]/\prod_{j\in R}\mathbf{u}[j]$ gives:

$$\mathrm{z}_1{}^2 \prod_{j\in T} \mathbf{u}[j]^2 \equiv \mathrm{z}_2{}^2 \prod_{j\in M\cup T} \mathbf{u}[j] \pmod{N}$$

From $\mathrm{c}_1 \neq \mathrm{c}_2$ it follows that $T \cup M \neq \emptyset$, and therefore by outputting $(\frac{\mathrm{z}_1}{\mathrm{z}_2}\prod_{j\in T}\mathbf{u}[j], M \cup T)$ the adversary B succeeds.

In the other case, when $\mathrm{c}_1 = \mathrm{c}_2$, it has to be $\mathrm{z}_1 \neq \mathrm{z}_2$ and the equation Equation (1) becomes $\mathrm{z}_1^2 \equiv \mathrm{z}_2^2 \bmod N$. By finding the gcd of N and $\mathrm{z}_1 - \mathrm{z}_2$, B can factorize N, and so it can compute the square root of $\mathbf{u}[1]$ modulo N. ∎

Proposition 4.3 and Theorem 4.1 imply that $\mathcal{H}$-$\mathcal{SMS}$ is collision-resistant under the SRPP assumption.

4.7 Additional functions

Okamoto's protocol [39] is StHVZK and can be shown to have special soundness if the discrete logarithm problem is hard relative to the underlying prime-order group generator, and hence we obtain a collision-resistant Σ-hash family $\mathcal{H}$-$\mathcal{Oka}$. The key has the form $(\langle G\rangle, p, g_1, g_2, X)$, where $g_1, g_2 \in G^*$ and $X \in G$, and $\mathsf{H}_{pk} : \mathbb{Z}_p \times (\mathbb{Z}_p \times \mathbb{Z}_p)$ is defined by $\mathsf{H}_{pk}(\mathrm{c}, (\mathrm{z}_1, \mathrm{z}_2)) = X^{\mathrm{c}} g_1^{\mathrm{z}_1} g_2^{\mathrm{z}_2}$. However, this hash function seems to offer no performance advantage over $\mathcal{H}$-$\mathcal{Sch}$. A pairing based identification protocol $\mathcal{HS}$, derived from the id-based signature scheme of [25], is noted in [5]. It is shown in [5] to have special soundness under concurrent attack assuming the hardness of the one more computational Diffie-Helman problem relative to an underlying prime-order bilinear group generator. The proof can be easily extended to show strong special soundness while relaxing the assumption to the hardness of the computational Diffie-Helman problem. $\mathcal{HS}$ can also be shown to be StHVZK and hence we obtain a Σ-hash family $\mathcal{H}$-$\mathcal{HS}$. The key has the form $(\langle G_1\rangle, \langle G_2\rangle, q, P, \langle \mathrm{e}\rangle, \alpha)$ where G_1 and G_2 are groups of prime order p; $\mathrm{e} : G_1 \times G_1 \to G_2$, is non-degenerate bilinear map; $P \in G_1^*$ and $\alpha \in G_2$. The function $\mathsf{H}_{pk} : \mathbb{Z}_p \times G_1 \to G_2$ is defined by $\mathsf{H}_{pk}(\mathrm{c}, \mathrm{z}) = \mathrm{e}(\mathrm{z}, P) \cdot \alpha^{\mathrm{c}}$. Due to the pairing, however, this hash function is slower than $\mathcal{H}$-$\mathcal{Sch}$.

5 Σ = chameleon

We move from examples of Σ-hash functions to a general property of the class, namely that any Σ-hash function is chameleon and vice-versa.

5.1 Definitions

A 2-input hash family $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{D}_1, \mathrm{D}_2, \mathrm{R})$ is said to be trapdoor if the following are true. First, there is a PT algorithm K, called the full key-generation algorithm, that on input 1^k returns a pair (K, T), and algorithm KG is equal to $\mathsf{K}^{(1)}$, meaning algorithm KG, on input 1^k, runs $\mathsf{K}(1^k)$ to get (K, T) and returns K. Second, there is a deterministic, PT algorithm, I, called the inversion algorithm, such that for all $(K, T) \in [\mathsf{K}(1^k)]$ and all $\mathrm{c}_1, \mathrm{c}_2 \in \mathrm{D}_1(K)$ and all $\mathrm{Y} \in \mathrm{R}(K)$ the map defined by $\mathrm{z} \to \mathit{I}(K, T, \mathrm{Y}, \mathrm{c}_1, \mathrm{z}, \mathrm{c}_2)$ is a bijection of $S^{\mathcal{H}}_{K,\mathrm{c}_1}(\mathrm{Y})$ to $S^{\mathcal{H}}_{K,\mathrm{c}_2}(\mathrm{Y})$, where $S^{\mathcal{H}}_{K,\mathrm{c}}(\mathrm{Y}) = \{\mathrm{z} \in \mathrm{D}_2(K) :$

$\mathsf{H}_K(\mathrm{C},\mathrm{Z}) = \mathrm{Y}\}$.[1] We say that $\mathcal{H}$ has the uniformity property if for all K and all $\mathrm{C} \in \mathrm{D}_1(K)$ it is the case that $\mathsf{H}_K(\mathrm{C},\cdot)$ is uniformly distributed over $\mathrm{R}(K)$ when regarded as a random variable over a random choice of its argument Z from $\mathrm{D}_2(K)$. We say that $\mathcal{H}$ is chameleon if it is trapdoor, collision-resistant and has the uniformity property.

The (standard) completeness requirement for a Σ-protocol $\mathcal{SP} = (\mathsf{K},\mathsf{P},\mathsf{V},\mathrm{CmSet},\mathrm{ChSet},\mathrm{RpSet})$ implies that from a secret key sk and challenge C, one can easily (in PT) compute the response Z, but only if one has the ephemeral secret key y underlying the commitment. To obtain chameleon hash functions from Σ-protocols we need the latter to satisfy a strong form of completeness which says that a response, distributed identically to the response of the real prover P, can be computed even without the ephemeral secret key so long as we have access to some accepting conversation. Formally a strong HVZK Σ-protocol $\mathcal{SP} = (\mathsf{K},\mathsf{P},\mathsf{V},\mathrm{CmSet},\mathrm{ChSet},\mathrm{RpSet})$ satisfies strong completeness if there is a deterministic PT algorithm $\mathcal{P}^*$ called the strong prover such that for all $(pk,sk) \in \left[\mathsf{K}(1^k)\right]$ and all $\mathrm{C}_1,\mathrm{C}_2 \in \mathrm{ChSet}(pk)$ and all $\mathrm{Y} \in \mathrm{CmSet}(pk)$ the map defined by $\mathrm{Z} \to \mathcal{P}^*(pk,sk,\mathrm{Y},\mathrm{C}_1,\mathrm{Z},\mathrm{C}_2)$ is a bijection of $S^{\mathcal{SP}}_{pk,\mathrm{C}_1}(\mathrm{Y})$ to $S^{\mathcal{SP}}_{pk,\mathrm{C}_2}(\mathrm{Y})$, where $S^{\mathcal{SP}}_{pk,\mathrm{C}}(\mathrm{Y}) = \{\mathrm{Z} \in \mathrm{RpSet}(pk) : \mathsf{StSim}(pk,\mathrm{C},\mathrm{Z}) = \mathrm{Y}\}$ where StSim is the strong HVZK simulator. We let $\boldsymbol{\Sigma}(\mathrm{sc})$ be the class of all Σ-protocols that have the strong completeness property.

5.2 Sigma is chameleon

The following implies that any Σ-hash is chameleon.

Theorem 5.1 Let $\mathcal{SP} = (\mathsf{K},\mathsf{P},\mathsf{V},\mathrm{CmSet},\mathrm{ChSet},\mathrm{RpSet}) \in \boldsymbol{\Sigma}(\mathrm{StHVZK}) \cap \boldsymbol{\Sigma}(\mathrm{sss}) \cap \boldsymbol{\Sigma}(\mathrm{sc})$ be a Σ-protocol. Then the Σ-hash family $\mathcal{H}\text{-}\mathcal{SP} = \boldsymbol{\Sigma}2\mathsf{H}(\mathcal{SP}) = (\mathsf{KG},\mathsf{H},\mathrm{ChSet},\mathrm{RpSet},\mathrm{CmSet})$ is chameleon.

Proof of Theorem 5.1: Theorem 4.1 implies that $\mathcal{H}\text{-}\mathcal{SP} = (\mathsf{KG},\mathsf{H},\mathrm{ChSet},\mathrm{RpSet},\mathrm{CmSet})$ is collision resistant. We now show that the strong HVZK property of $\mathcal{SP}$ implies uniformity of $\mathcal{H}\text{-}\mathcal{SP}$. Fix $(pk,sk) \in \left[\mathsf{K}(1^k)\right]$ and also fix $\mathrm{C} \in \mathrm{ChSet}(pk)$. We want to show that $\mathsf{H}_{pk}(\mathrm{C},\cdot) = \mathsf{StSim}(pk,\mathrm{C},\cdot)$ is uniformly distributed over $\mathrm{CmSet}(pk)$ when its argument is drawn at random from $\mathrm{RpSet}(pk)$. Consider the games of Figure 7. Let D be any (computationally unbounded) adversary. Then it suffices to show that

$$\Pr\left[\mathrm{S}^D \Rightarrow 1\right] = \Pr\left[\mathrm{T}^D \Rightarrow 1\right]$$

where "$G^D \Rightarrow 1$" denotes the event that D outputs 1 on input the output of game G, and the probability is over the coins of G and D. But this follows because

$$\Pr\left[\mathrm{S}^D \Rightarrow 1\right] = \Pr\left[\mathrm{R}^D \Rightarrow 1\right] \quad (2)$$

$$\Pr\left[\mathrm{R}^D \Rightarrow 1\right] = \Pr\left[\mathrm{T}^D \Rightarrow 1\right], \quad (3)$$

where game T is also in Figure 7. Equation (2) is true by the strong HVZK property. (The real and simulated conversation transcripts are equally distributed, and hence continue to be so conditioned on a particular challenge.) Equation (3) is true because our definition of strong HVZK required that a commitment generated by the prover is uniformly distributed over $\mathrm{CmSet}(pk)$.

[1] Krawczyk and Rabin [31] only require that $\mathsf{H}_K(\mathrm{C}_2, I(K,T,\mathsf{H}_K(\mathrm{C}_1,\mathrm{Z}_1),\mathrm{C}_1,\mathrm{Z}_1,\mathrm{C}_2)) = \mathsf{H}_K(\mathrm{C}_1,\mathrm{Z}_1)$ for all $\mathrm{C}_1,\mathrm{C}_2 \in \mathrm{ChSet}(K)$ and all $\mathrm{Z} \in \mathrm{RpSet}(K)$. Shamir and Tauman require a stronger condition that is essentially a computational version of ours. It seems to us that the non-transferability of chameleon signatures required in [31] requires the hash function to meet one of these stronger conditions.

Game R	Game S	Game T
$(\mathrm{Y}, \mathrm{y}) \leftarrow\!\!\$\ \mathsf{P}(pk, sk)$	$\mathrm{z} \leftarrow\!\!\$\ \mathrm{RpSet}(pk)$	$\mathrm{Y} \leftarrow\!\!\$\ \mathrm{CmSet}(pk)$
$\mathrm{z} \leftarrow\!\!\$\ \mathsf{P}(pk, sk, \mathrm{c}, \mathrm{y})$	$\mathrm{Y} \leftarrow\!\!\$\ \mathsf{StSim}(pk, \mathrm{c}, \mathrm{z})$	Return Y
Return Y	Return Y	

Figure 7: Games for proof of uniformity of $\mathcal{H}$-$\mathcal{SP}$ in proof of Theorem 5.1. Here pk, sk, c are fixed.

$\underline{\mathsf{P}(pk, sk)}$	$\underline{\mathsf{P}(pk, sk, \mathrm{c}_2, \mathrm{y})}$
$\mathrm{c}_1 \leftarrow\!\!\$\ \mathrm{ChSet}(pk)$	$(\mathrm{Y}, \mathrm{c}_1, \mathrm{z}_1) \leftarrow \mathrm{y}$
$\mathrm{z}_1 \leftarrow\!\!\$\ \mathrm{RpSet}(pk)$	$\mathrm{z}_2 \leftarrow \mathcal{I}(pk, sk, \mathrm{Y}, \mathrm{c}_1, \mathrm{z}_1, \mathrm{c}_2)$
$\mathrm{Y} \leftarrow \mathsf{H}_{pk}(\mathrm{c}, \mathrm{z})$	Return z_2
$\mathrm{y} \leftarrow (\mathrm{Y}, \mathrm{c}_1, \mathrm{z}_1)$	
Return (Y, y)	

Figure 8: Prover algorithm for the proof of Theorem 5.2. In line 2 of the second column, $(\mathrm{Y}, \mathrm{c}_1, \mathrm{z}_1) \leftarrow \mathrm{y}$ means we parse y as shown.

To show that $\mathcal{H}$-$\mathcal{SP}$ is trapdoor we need to exhibit the full key-generation algorithm and the inversion algorithm. The full key-generation algorithm is simply the key-generation algorithm K of $\mathcal{SP}$, so that the trapdoor is the secret key of the protocol. The inversion algorithm is simply the strong prover from the strong completeness condition. That the trapdoor condition is met is a tautology, since the set $S^{\mathcal{H}}_{pk,\mathrm{c}}(\mathrm{Y})$ is exactly the set $S^{\mathcal{SP}}_{pk,\mathrm{c}}(\mathrm{Y})$. ▌

As a consequence, we obtain the following new chameleon hash functions: $\mathcal{H}$-$\mathcal{GQ}$, $\mathcal{H}$-$\mathcal{SFS}$, $\mathcal{H}$-$\mathcal{SMS}$, $\mathcal{H}$-$\mathcal{OKA}$, $\mathcal{H}$-$\mathcal{HS}$. ($\mathcal{H}$-$\mathcal{Sch}$ was already known to be chameleon [31].) This yields numerous new and more efficient instantiations of on-line/off-line signatures [47], chameleon signatures [31] and designated-verifier signatures [29, 50]. It also provides new and more efficient ways to turn weakly-secure signatures into fully-secure ones that can improve the performance of schemes like [27].

5.3 Chameleon is Sigma

We also prove the converse. The following theorem says that any chameleon hash family is a Σ-hash family, meaning the result of applying our Σ2H transform to some Σ-protocol.

Theorem 5.2 Let $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{ChSet}, \mathrm{RpSet}, \mathrm{CmSet})$ be a family of chameleon hash functions. Then there is a Σ-protocol $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathrm{CmSet}, \mathrm{ChSet}, \mathrm{RpSet}) \in \Sigma(\mathrm{StHVZK}) \cap \Sigma(\mathrm{sss}) \cap \Sigma(\mathrm{sc})$ such that $\mathcal{H} = \Sigma 2\mathsf{H}(\mathcal{SP})$ is the Σ-hash family corresponding to $\mathcal{SP}$.

Proof of Theorem 5.2: Since $\mathcal{H}$ is trapdoor, it has a full key-generation algorithm K and an inversion algorithm $\mathcal{I}$. Let the former be the key-generation algorithm of $\mathcal{SP}$. Now we define the prover algorithm as shown in Figure 8. Define the verifier V on input $pk, \mathrm{Y} \| \mathrm{c} \| \mathrm{z}$ to output 1 if $\mathsf{H}_{pk}(\mathrm{c}, \mathrm{z}) = \mathrm{Y}$ and $\mathrm{Y} \in \mathrm{CmSet}(pk)$ and $\mathrm{c} \in \mathrm{ChSet}(pk)$ and $\mathrm{z} \in \mathrm{RpSet}(pk)$, and 0 otherwise. We now need to show that $\mathcal{SP}$ satisfies strong HVZK, strong special soundness, and strong completeness. (We need to also show that $\mathcal{SP}$ satisfies completeness, but this is implied by strong completeness.)

Let StSim be defined by $\mathsf{StSim}(pk, \mathrm{c}, \mathrm{z}) = \mathsf{H}_{pk}(\mathrm{c}, \mathrm{z})$. We now show this is a strong HVZK simulator. Fix $(pk, sk) \in \mathsf{K}(1^k)$ and consider the games of Figure 9. Game R generates real protocol transcripts

Game R	Game S	Game T	Game U
$\mathrm{C}_1 \leftarrow\!\!\$\ \mathrm{ChSet}(pk)$	$\mathrm{C} \leftarrow\!\!\$\ \mathrm{ChSet}(pk)$	$\mathrm{C}_1 \leftarrow\!\!\$\ \mathrm{ChSet}(pk)$	$\mathrm{Y} \leftarrow\!\!\$\ \mathrm{CmSet}(pk)$
$\mathrm{Z}_1 \leftarrow\!\!\$\ \mathrm{RpSet}(pk)$	$\mathrm{Z} \leftarrow\!\!\$\ \mathrm{RpSet}(pk)$	$\mathrm{Y} \leftarrow\!\!\$\ \mathrm{CmSet}(pk)$	$\mathrm{C}_2 \leftarrow\!\!\$\ \mathrm{ChSet}(pk)$
$\mathrm{Y} \leftarrow \mathsf{H}_{pk}(\mathrm{C}, \mathrm{Z})$	$\mathrm{Y} \leftarrow \mathsf{H}_{pk}(\mathrm{C}, \mathrm{Z})$	$\mathrm{Z}_1 \leftarrow\!\!\$\ S^{\mathcal{H}}_{pk,\mathrm{C}_1}(\mathrm{Y})$	$\mathrm{Z}_2 \leftarrow\!\!\$\ S^{\mathcal{H}}_{pk,\mathrm{C}_2}(\mathrm{Y})$
$\mathrm{C}_2 \leftarrow\!\!\$\ \mathrm{ChSet}(pk)$	Return $(pk, \mathrm{Y}\|\mathrm{C}\|\mathrm{Z})$	$\mathrm{C}_2 \leftarrow\!\!\$\ \mathrm{ChSet}(pk)$	Return $(pk, \mathrm{Y}\|\mathrm{C}_2\|\mathrm{Z}_2)$
$\mathrm{Z}_2 \leftarrow I(pk, sk, \mathrm{Y}, \mathrm{C}_1, \mathrm{Z}_1, \mathrm{C}_2)$		$\mathrm{Z}_2 \leftarrow I(pk, sk, \mathrm{Y}, \mathrm{C}_1, \mathrm{Z}_1, \mathrm{C}_2)$	
Return $(pk, \mathrm{Y}\|\mathrm{C}_2\|\mathrm{Z}_2)$		Return $(pk, \mathrm{Y}\|\mathrm{C}_2\|\mathrm{Z}_2)$	

Figure 9: Games for proof of strong HVZK in proof of Theorem 5.2. Here (pk, sk) are fixed.

based on the prover algorithm of Figure 8 while S generates a simulated transcript based on StSim. We want to show that

$$\Pr\left[\mathrm{R}^D \Rightarrow 1\right] = \Pr\left[\mathrm{S}^D \Rightarrow 1\right] \tag{4}$$

for any (computationally unbounded) adversary D. But the uniformity property implies that

$$\Pr\left[\mathrm{R}^D \Rightarrow 1\right] = \Pr\left[\mathrm{T}^D \Rightarrow 1\right].$$

On the other hand, by the trapdoor property we have

$$\Pr\left[\mathrm{T}^D \Rightarrow 1\right] = \Pr\left[\mathrm{U}^D \Rightarrow 1\right]$$

Re-applying uniformity we have

$$\Pr\left[\mathrm{U}^D \Rightarrow 1\right] = \Pr\left[\mathrm{S}^D \Rightarrow 1\right]$$

and so we have equation Equation (4).

The collision-resistance of $\mathcal{H}$ directly implies strong special soundness of $\mathcal{SP}$. Also, the trapdoor property of $\mathcal{H}$ implies strong completeness of $\mathcal{SP}$ by simply letting the strong prover for the strong completeness condition be the inversion algorithm of the trapdoor condition. Again, the required conditions are met simply because the sets $S^{\mathcal{H}}_{pk,\mathrm{C}}(\mathrm{Y})$ and $S^{\mathcal{SP}}_{pk,\mathrm{C}}(\mathrm{Y})$ are the same. ∎

Applying this to known chameleon-hash functions like $\mathcal{H}$-$\mathcal{DL}$ [17, 31] and $\mathcal{H}$-$\mathcal{ST}$ [47] yields new Σ-protocols and hence new identification schemes and, via [19, 16], new commitment schemes.

6 Σ-hash practice and performance

In this section we cover practical issues related to Σ-hash functions, including performance, performance comparison with existing constructions and implementation results.

6.1 Extending the domain

A Σ-hash family $\mathcal{H}$ as defined above is actually a (keyed) compression function since the domain is relatively small. In practice however we need to hash messages of long and variable length. This would not at first appear to be much of a problem since we should be able to do MD iteration [18, 35]. In fact this is essentially true but one has to be careful about a few things. What one would naturally like to do is use the second argument to H_{pk} as the chaining variable. But this requires that outputs of the compression function can be regarded as chaining values, meaning $\mathrm{CmSet}(pk)$ be a subset of $\mathrm{RpSet}(pk)$. Sometimes this is true, as for $\mathcal{H}$-$\mathcal{GQ}$, which in this way lends itself

Σ-hash	w	KB/s	space
$\mathcal{H}$-$\mathcal{SFS}$	0	30.85	n/a
$\mathcal{H}$-$\mathcal{SFS}$	4	67.41	2048
$\mathcal{H}$-$\mathcal{SFS}$	8	118.1	16384
$\mathcal{H}$-$\mathcal{SMS}$	0	914.3	n/a

Table 1: Implementation results. Here w is the "width" parameter determining pre-computation and the space is the number of group elements that need to be stored.

---

easily and naturally to MD iteration. But in the case of $\mathcal{SFS}$ and $\mathcal{SMS}$ we have $\mathrm{CmSet}((N, l, \mathbf{u})) = \mathbb{Z}_N^* \subsetneqq \mathbb{Z}_N^+ = \mathrm{RpSet}((N, l, \mathbf{u}))$. In Appendix A we show how to resolve these problems by appropriate "embeddings" that effectively allow the second input of the compression function to be used as a chaining variable at the cost of 1 bit in throughput and in particular allows us to run any of our Σ-hash functions in MD mode. We won't detail the general transform here, but it is instructive to describe the modified compression function. The public key has the form $(N, l, \mathbf{u}, v)$ where $N, l, \mathbf{u}$ are as before and $v \in \mathrm{QR}(N)$, and $\mathsf{H}_{pk} : \{0,1\}^l \times \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is defined by

$$\mathsf{H}_{pk}(\mathrm{C}, \mathrm{Z}) = \mathbf{u}^{\mathrm{C}} \cdot \mathrm{Z}^2 \cdot v^{f_N(\mathrm{Z})} \bmod N, \tag{5}$$

where $f_N(\mathrm{Z}) = 0$ if $\mathrm{Z} \in \mathbb{Z}_N^+$ and 1 otherwise. It can be shown that this modified function is also a Σ-hash, meaning the result of applying Σ2H to a suitably modified version of the original Σ-protocol that retains the sss, StHVZK and sc properties of the original. But now $\mathrm{CmSet}((N, l, \mathbf{u}, v)) = \mathbb{Z}_N^* = \mathrm{RpSet}((N, l, \mathbf{u}, v))$ so MD-iteration is possible.

6.2 Metrics

We measure performance of a hash function in terms of rate, which we define as the average number of bits hashed per group operations. (By "average" we mean when the data is random.) In this measure, an exponentiation $a \mapsto A^a$ costs $1.5n$ group operations and a two-fold multi-exponentiation $a, b \mapsto A^a B^b$ costs $1.75n$ group operations where n is the length of a and also of b. We will use these estimates extensively below. We can consider two modes of operation of a given Σ-hash function $\mathcal{H}$-$\mathcal{SP}$, namely compression and MD. In the first case the data to be hashed by H_{pk} is the full input C, Z, while in the second case it is only C. (The second input is the chaining variable which is not part of the data.) The rate in MD mode is lower than in compression mode for most hash functions. ($\mathcal{SFS}$ is an interesting exception.) Compression mode is relevant when the function is being used as a chameleon hash, since the data can then be compressed with a standard (merely collision-resistant) hash function such as SHA-1 before applying the Σ-hash [31, Lemma 1]. MD mode is relevant when one wants to avoid conventional hash functions and get the full provable guarantees of the Σ-hash by using it alone. Our performance evaluations will consider MD mode.

6.3 Performance of Σ-hash functions

$\mathcal{H}$-$\mathcal{Sch}$ and $\mathcal{H}$-$\mathcal{GQ}$ can be computed with one two-fold multi-exponentiation so that they use 1.75 group operations per bit of data (in MD mode). We now turn to $\mathcal{H}$-$\mathcal{SFS}$. Since we are considering MD mode performance we refer to the MD-compatible version of the function from Equation (5). (But

in fact performance is hardly affected by the modification.) On the average about half the bits of C are 1 so $\mathcal{H}$-$\mathcal{SFS}$ comes in at about 0.5 modular multiplications per bit. This explains the claim of Figure 1 in regard to $\mathcal{H}$-$\mathcal{SFS}$ without pre-computation. Now we look at how pre-computation speeds it up, using a block size of $l = 512$ (the same as MD5 and SHA-1) for illustration. The method is obvious. Pick a "width" w that divides l and let $t = l/w$. Letting $pk = (N, l, \mathbf{u}, v)$ denote the public key, pre-compute and store the table T with entries

$$T[i, x] = \prod_{j=1}^{w} \mathbf{u}[(i-1)w + j]^x \bmod N \quad (1 \le i \le t,\ x \in \{0, \ldots 2^w - 1\})$$

The size of the table is $t2^w = l2^w/w$ group elements. Now computing $\mathcal{H}$-$\mathcal{SFS}$ takes $t + 2 = 2 + l/w$ multiplications since

$$\mathsf{H}_{pk}(\mathrm{C}, \mathrm{Z}) = \left(\prod_{i=1}^{t} T[i, x_i]\right) \cdot \mathrm{Z}^2 \cdot v^{f_N(\mathrm{Z})} \bmod N,$$

where x_i is the integer with binary representation $\mathrm{C}[(i-1)w + 1] \ldots c[iw]$ $(1 \le i \le t)$. The number of group operations per bit is thus $[2 + l/w]/l \approx 1/w$, meaning the rate is w. Figure 1 showed the storage and this rate for $w = 4$ and $w = 8$.

Analytical assessment of the performance of $\mathcal{H}$-$\mathcal{SMS}$ is difficult, but we have implemented both it and (for comparison) $\mathcal{H}$-$\mathcal{SFS}$. The implementation used a 1024 bit modulus and (for MD mode) a 512 bit block size. Table 1 shows that $\mathcal{H}$-$\mathcal{SMS}$ is about 30 times faster than the basic (no pre-computation) version of $\mathcal{H}$-$\mathcal{SFS}$. The gap drops to a factor of 15 and 7.5 when compared with the $w = 4$ and $w = 8$ pre-computation levels of $\mathcal{H}$-$\mathcal{SFS}$, respectively. Note that $\mathcal{H}$-$\mathcal{SMS}$ here is without pre-computation. (The latter does not seem to help it much.) These implementation results are on a Dual Pentium IV, 3.2GHz machine, running Linux kernel 2.6 and using the gmp library [22].

6.4 Comparisons

We now assess performance of previous schemes, justifying claims in Section 1. Damgård [17] shows how to construct collision-resistant hash functions from claw-free permutations [23]. Of various factoring-based instantiations of his construction, the one of [23, 31], which we denote $\mathcal{H}$-$\mathcal{Da}$, seems to be the most efficient. The key is a modulus N product of two primes, one congruent to 3 mod 8 and the other to 7 mod 8, and the hash function $\mathsf{H}_N : \{0,1\}^l \times \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is defined by $\mathsf{H}_N(m, r) = 4^m \cdot r^s \bmod N$ where $s = 2^l$. Since multiplying by 4 is cheap, we view it as free and the cost is then one multiplication per bit, meaning $\mathcal{H}$-$\mathcal{SFS}$ is twice as fast. But pre-computation does not help $\mathcal{H}$-$\mathcal{Da}$ since r is not fixed, and the gap in rates increases as we allow pre-computation for $\mathcal{H}$-$\mathcal{SFS}$ as shown in Figure 1.

The key of Shamir and Tauman's [47] hash function is a modulus N and an $a \in \mathbb{Z}_N^*$. With a 1024 bit modulus the chaining variable needs to be 1024 bits as well, so that with a 512 bit block size the function would take a $512 + 1024$ bit input, regard it as an integer s, and return $a^s \bmod N$. The computation takes 1.5 multiplications per bit of the full input, which is $1.5 \cdot (1024 + 512)/512 = 4.5$ per bit of data, meaning the rate is $1/4.5 \approx 0.22$ as claimed in Figure 1. Since a is fixed, one can use the standard pre-computation methods for exponentiation. For any v dividing $1024 + 512 = 1536$, the computation takes $1536/v$ multiplications with a table of $2^v \cdot 1536/v$ group elements. Note that per data bit the rate is $512/(1536/v) = v/3$. To compare to $\mathcal{H}$-$\mathcal{SFS}$ we need to choose parameters so that the storage for the two is about the same, meaning $2^w(512/w) \approx 2^v(1536/v)$. This yields $v = 1$ for $w = 4$ and $v = 6$ for $w = 8$. This explains the rates shown in Figure 1.

7 Improvements to VSH

The performance of a hash function on short inputs is important in practice. (For example, a significant fraction of Internet traffic consists of short packets.) We present a variant VSH^* of VSH that is up to 5 times faster in this context while remaining proven-secure under the same assumption as VSH. The improvement stems from VSH^*, unlike VSH, having a collision-resistant compression function.

BACKGROUND. The key of Contini, Lenstra and Steinfeld's VSH function [14] is a modulus N product of two primes. The VSH compression function $\mathsf{vsh}_N : \{0,1\}^l \times \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ is defined by

$$\mathsf{vsh}_N(\mathrm{C}, \mathrm{Z}) = \mathrm{Z}^2 \cdot \prod_{i=1}^{l} p_i^{\mathrm{C}[i]} \bmod N,$$

where p_i is the i-th prime and $\mathrm{C}[i]$ is the i-th bit of C. The hash function VSH is obtained by MD-iteration of vsh with initial vector 1. A curious feature of VSH is that the compression function is *not* collision-resistant. Indeed, $\mathsf{vsh}_N(c, z) = \mathsf{vsh}_N(c, N - z)$ for any $c \in \{0,1\}^l$ and $z \in \mathbb{Z}_N^*$. Nonetheless, it is shown in [14] that the hash function VSH is collision-resistant based on the VSSR assumption. The latter states that given N, l it is hard to find $x \in \mathbb{Z}_N^*$ and integers $e_1, \ldots, e_l$, not all even, such that $x^2 \equiv p_1^{e_1} \cdot \ldots \cdot p_l^{e_l} \pmod N$. The proof makes crucial use of the fact that the initial vector is set to 1.

VSH^*. We alter the compression function of VSH so that it becomes (provably) collision-resistant and then define VSH^* by MD iteration with the initial vector being part of the data to be hashed. The first application of the compression function thus consumes much more (1024 bits more for a 1024 bit modulus, for example) of the input, resulting in significantly improved rate for the important practical case of hashing short messages. For example, the implementation results of Table 2 show speed increases of a factor of 5 over VSH when hashing 1024 bit messages. Performance for long messages is the same as for VSH. VSH^* and its compression function vsh^* are provably collision-resistant under the same VSSR assumption as VSH.

The inspiration comes from $\mathcal{H}$-$\mathcal{SMS}$ which we notice is very similar to vsh but, unlike the latter, is collision-resistant. The difference is that in $\mathcal{H}$-$\mathcal{SMS}$ the primes $\mathbf{u}[1], \ldots, \mathbf{u}[l], v$ —referring to the MD-compatible version of the function from Equation (5)— are quadratic residues. But this turns out to be important for the completeness of the Σ-protocol rather than for collision-resistance. This leads to the compression function $\mathsf{vsh}_N^* : \{0,1\}^l \times \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ defined by

$$\mathsf{vsh}_N^*(\mathrm{C}, \mathrm{Z}) = \left(\prod_{i=1}^{l} p_i^{\mathrm{C}[i]}\right) \cdot p_{l+1}^{f_N(\mathrm{Z})} \cdot \mathrm{Z}^2 \bmod N,$$

where $f_N(\mathrm{Z}) = 0$ if $\mathrm{Z} \in Z_N^+$ and 1 otherwise, p_i is the i-th prime and $\mathrm{C}[i]$ is the i-th bit of C. As a check notice that $\mathsf{vsh}_N^*(\mathrm{C}, \mathrm{Z})$ is unlikely to equal $\mathsf{vsh}_N^*(\mathrm{C}, N - \mathrm{Z})$ because $f_N(\mathrm{Z}) \neq f_N(N - \mathrm{Z})$, meaning the attack showing vsh is not collision-resistant does not apply. Of course this is not the only possible attack, but the proof of strong special soundness of $\mathcal{SMS}$ Proposition 4.3 can be adapted to show that vsh^* is collision-resistant under the VSSR assumption. Finally VSH^* is obtained by MD iteration of vsh^* but with the initial vector being the first $k - 1$ bits of the input. For MD-strengthening, the standard padding method of SHA-1 is used.

The implementation results given in Table 2 were again obtained on a Pentium IV, 3 GHz machine using the gmp library [22]. We set the block size to 128 for both functions and considered hashing a 1024 bit input. In this case (even taking into account the increase in length due to MD strengthening) VSH^* needs 1 application of its compression function. On the other hand VSH (with their own form of strengthening) needs 9. The implementation shows that VSH^* is 5.6 times faster. We need to add that our implementations (unlike those of [14]) are not optimized, but our goal was more to assess the comparative than the absolute performance of these hash functions, and this is achieved because both

Hash Function	block size	input size	Iterations	Avg. time
VSH	128	8×128	9	$140\mu s$
VSH*	128	8×128	1	$25\mu s$

Table 2: The size of the modulus used here is 1024. The block and the input size are given in bits.

are tested on the same platform.

Acknowledgments

We thank the Journal of Cryptology referees for their valuable comments and careful reading of the paper.

References

[1] Elena Andreeva, Gregory Neven, Bart Preneel, and Thomas Shrimpton. Seven-property-preserving iterated hashing: ROX. In Kaoru Kurosawa, editor, *ASIACRYPT 2007*, volume 4833 of *LNCS*, pages 130–146, Kuching, Malaysia, December 2–6, 2007. Springer, Berlin, Germany.

[2] Giuseppe Ateniese and Breno de Medeiros. Identity-based chameleon hash and applications. In Ari Juels, editor, *FC 2004*, volume 3110 of *LNCS*, pages 164–180, Key West, USA, February 9–12, 2004. Springer, Berlin, Germany.

[3] Giuseppe Ateniese and Breno de Medeiros. On the key exposure problem in chameleon hashes. In Carlo Blundo and Stelvio Cimato, editors, *SCN 04*, volume 3352 of *LNCS*, pages 165–179, Amalfi, Italy, September 8–10, 2004. Springer, Berlin, Germany.

[4] Mihir Bellare and Daniele Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In Walter Fumy, editor, *EUROCRYPT'97*, volume 1233 of *LNCS*, pages 163–192, Konstanz, Germany, May 11–15, 1997. Springer, Berlin, Germany.

[5] Mihir Bellare, Chanathip Namprempre, and Gregory Neven. Security proofs for identity-based identification and signature schemes. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 268–286, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

[6] Mihir Bellare and Thomas Ristenpart. Multi-property-preserving hash domain extension and the EMD transform. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284 of *LNCS*, Shanghai, China, December 3–7, 2006. Springer, Berlin, Germany.

[7] Mihir Bellare and Thomas Ristenpart. Hash functions in the dedicated-key setting: Design choices and MPP transforms. In Lars Arge, Christian Cachin, Tomasz Jurdzinski, and Andrzej Tarlecki, editors, *ICALP 2007*, volume 4596 of *LNCS*, pages 399–410, Wroclaw, Poland, July 9–13, 2007. Springer, Berlin, Germany.

[8] Mihir Bellare and Todor Ristov. Hash functions from sigma protocols and improvements to VSH. In Josef Pieprzyk, editor, *ASIACRYPT 2008*, volume 5350 of *LNCS*, pages 125–142, Melbourne, Australia, December 7–11, 2008. Springer, Berlin, Germany.

[9] Mihir Bellare and Sarah Shoup. Two-tier signatures, strongly unforgeable signatures, and Fiat-Shamir without random oracles. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *PKC 2007*, volume 4450 of *LNCS*, pages 201–216, Beijing, China, April 16–20, 2007. Springer, Berlin, Germany.

[10] Thomas Beth. Efficient zero-knowledge identification scheme for smart cards. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 77–86, Davos, Switzerland, May 25–27, 1988. Springer, Berlin, Germany.

[11] Dan Boneh and Xavier Boyen. Short signatures without random oracles. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 56–73, Interlaken, Switzerland, May 2–6, 2004. Springer, Berlin, Germany.

[12] Denis Xavier Charles, Kristin E. Lauter, and Eyal Z. Goren. Cryptographic hash functions from expander graphs. *Journal of Cryptology*, 22(1):93–113, January 2009.

[13] David Chaum, Eugéne van Heijst, and Birgit Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In Joan Feigenbaum, editor, *CRYPTO'91*, volume 576 of *LNCS*, pages 470–484, Santa Barbara, CA, USA, August 11–15, 1991. Springer, Berlin, Germany.

[14] Scott Contini, Arjen K. Lenstra, and Ron Steinfeld. VSH, an efficient and provable collision-resistant hash function. In Serge Vaudenay, editor, *EUROCRYPT 2006*, volume 4004 of *LNCS*, pages 165–182, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Berlin, Germany.

[15] Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya. Merkle-Damgård revisited: How to construct a hash function. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 430–448, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.

[16] Ronald Cramer, Ivan Damgrd, and Philip MacKenzie. Efficient zero-knowledge proofs of knowledge without intractability assumptions. In David Naccache and Pascal Paillier, editors, *PKC 2002*, volume 2274 of *LNCS*, pages 354–372, Paris, France, February 12–14, 2002. Springer, Berlin, Germany.

[17] Ivan Damgård. Collision free hash functions and public key signature schemes. In C. G. Günther, editor, *EUROCRYPT'88*, volume 330 of *LNCS*, pages 203–216, Davos, Switzerland, May 25–27, 1988. Springer, Berlin, Germany.

[18] Ivan Damgård. A design principle for hash functions. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 17–27, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Berlin, Germany.

[19] Ivan Damgård. On the existence of bit commitment schemes and zero-knowledge proofs. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 17–27, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Berlin, Germany.

[20] Shimon Even, Oded Goldreich, and Silvio Micali. On-line/off-line digital signatures. *Journal of Cryptology*, 9(1):35–67, 1996.

[21] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. In Andrew M. Odlyzko, editor, *CRYPTO'86*, volume 263 of *LNCS*, pages 186–194, Santa Barbara, CA, USA, August 1987. Springer, Berlin, Germany.

[22] The gnu mp bignum library. `http://gmplib.org/`.

[23] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal on Computing*, 17(2):281–308, April 1988.

[24] Lous C. Guillou and Jean-Jacques Quisquater. A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 216–231, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Berlin, Germany.

[25] Florian Hess. The GHS attack revisited. In Eli Biham, editor, *EUROCRYPT 2003*, volume 2656 of *LNCS*, pages 374–387, Warsaw, Poland, May 4–8, 2003. Springer, Berlin, Germany.

[26] Susan Hohenberger and Brent Waters. Realizing hash-and-sign signatures under standard assumptions. In Antoine Joux, editor, *EUROCRYPT 2009*, volume 5479 of *LNCS*, pages 333–350, Cologne, Germany, April 26–30, 2009. Springer, Berlin, Germany.

[27] Susan Hohenberger and Brent Waters. Short and stateless signatures from the RSA assumption. In Shai Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 654–670, Santa Barbara, CA, USA, August 16–20, 2009. Springer, Berlin, Germany.

[28] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 445–456, Cambridge, MA, USA, February 10–12, 2005. Springer, Berlin, Germany.

[29] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In Ueli M. Maurer, editor, *EUROCRYPT'96*, volume 1070 of *LNCS*, pages 143–154, Saragossa, Spain, May 12–16, 1996. Springer, Berlin, Germany.

[30] J. Jonsson and B.Kaliski. Public-key cryptography standards (PKCS) #1: RSA cryptography, specifications version 2.1., February 2003. In Internet RFC 3447.

[31] Hugo Krawczyk and Tal Rabin. Chameleon signatures. In *NDSS 2000*, San Diego, California, USA, February 2–4, 2000. The Internet Society.

[32] Kaoru Kurosawa and Swee-Huay Heng. The power of identification schemes. In Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin, editors, *PKC 2006*, volume 3958 of *LNCS*, pages 364–377, New York, NY, USA, April 24–26, 2006. Springer, Berlin, Germany.

[33] Vadim Lyubashevsky and Daniele Micciancio. Generalized compact Knapsacks are collision resistant. In Michele Bugliesi, Bart Preneel, Vladimiro Sassone, and Ingo Wegener, editors, *ICALP 2006, Part II*, volume 4052 of *LNCS*, pages 144–155, Venice, Italy, July 10–14, 2006. Springer, Berlin, Germany.

[34] Vadim Lyubashevsky, Daniele Micciancio, Chris Peikert, and Alon Rosen. SWIFFT: A modest proposal for FFT hashing. In Kaisa Nyberg, editor, *FSE 2008*, volume 5086 of *LNCS*, pages 54–72, Lausanne, Switzerland, February 10–13, 2008. Springer, Berlin, Germany.

[35] Ralph C. Merkle. A certified digital signature. In Gilles Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 218–238, Santa Barbara, CA, USA, August 20–24, 1990. Springer, Berlin, Germany.

[36] Silvio Micali and Adi Shamir. An improvement of the Fiat-Shamir identification and signature scheme. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 244–248, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Berlin, Germany.

[37] M. Naor. Bit commitment using pseudorandomness. *Journal of Cryptology*, 4(2):151–158, 1991.

[38] Kazuo Ohta and Tatsuaki Okamoto. A modification of the Fiat-Shamir scheme. In Shafi Goldwasser, editor, *CRYPTO'88*, volume 403 of *LNCS*, pages 232–243, Santa Barbara, CA, USA, August 21–25, 1990. Springer, Berlin, Germany.

[39] T. Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In Douglas R. Stinson, editor, *CRYPTO'93*, volume 773 of *LNCS*, pages 149–162, Santa Barbara, CA, USA, August 22–26, 1994. Springer, Berlin, Germany.

[40] H. Ong and Claus-Peter Schnorr. Fast signature generation with a Fiat-Shamir like scheme. In Ivan Damgård, editor, *EUROCRYPT'90*, volume 473 of *LNCS*, pages 432–440, Aarhus, Denmark, May 21–24, 1990. Springer, Berlin, Germany.

[41] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 223–238, Prague, Czech Republic, May 2–6, 1999. Springer, Berlin, Germany.

[42] C. Peikert and A. Rosen. Efficient collision-resistant hashing from. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 145–166, Cambridge, MA, USA, February 10–12, 2005. Springer, Berlin, Germany.

[43] Christophe Petit, Kristin Lauter, and Jean-Jacques Quisquater. Full cryptanalysis of LPS and Morgenstern hash functions. In Rafail Ostrovsky, Roberto De Prisco, and Ivan Visconti, editors, *SCN 08*, volume 5229 of *LNCS*, pages 263–277, Amalfi, Italy, September 10–12, 2008. Springer, Berlin, Germany.

[44] Christophe Petit, N. Veyrat-Charvillon, and Jean-Jacques Quisquater. Efficiency and pseudorandomness of a variant of Zémor-Tillich hash function. In *Electronics, Circuits and Systems, ICECS 2008*, pages 906–909. IEEE, September 2008.

[45] Ryuichi Sakai, Kiyoshi Ohgishi, and Masao Kasahara. Cryptosystems based on pairing. In *SCIS 2000*, Okinawa, Japan, January 2000.

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

[47] A. Shamir and Y. Tauman. Improved online/offline signature schemes. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 355–367, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.

[48] Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 355–367, Santa Barbara, CA, USA, August 19–23, 2001. Springer, Berlin, Germany.

[49] Daniel R. Simon. Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In Kaisa Nyberg, editor, *EUROCRYPT'98*, volume 1403 of *LNCS*, pages 334–345, Espoo, Finland, May 31 – June 4, 1998. Springer, Berlin, Germany.

[50] R. Steinfeld, H. Wang, and J. Pieprzyk. Efficient extension of standard Schnorr/RSA signatures into universal designated-verifier signatures. In Feng Bao, Robert Deng, and Jianying Zhou, editors, *PKC 2004*, volume 2947 of *LNCS*, pages 86–100, Singapore, March 1–4, 2004. Springer, Berlin, Germany.

[51] Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In Masayuki Abe, editor, *CT-RSA 2007*, volume 4377 of *LNCS*, pages 357–371, San Francisco, CA, USA, February 5–9, 2007. Springer, Berlin, Germany.

[52] J.-P. Tillich and G. Zemor. Collisions for the LPS expander graph hash function. In Nigel P. Smart, editor, *EUROCRYPT 2008*, volume 4965 of *LNCS*, Istanbul, Turkey, April 13–17, 2008. Springer, Berlin, Germany.

[53] Jean-Pierre Tillich and Gilles Zémor. Hashing with SL_2. In Yvo Desmedt, editor, *CRYPTO'94*, volume 839 of *LNCS*, pages 40–49, Santa Barbara, CA, USA, August 21–25, 1994. Springer, Berlin, Germany.

[54] X. Wang, Y.L. Yin, and H. Yu. Finding collisions in the full SHA-1. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 17–36, Santa Barbara, CA, USA, August 14–18, 2005. Springer, Berlin, Germany.

[55] X. Wang, Y.L. Yin, and H. Yu. How to break MD5 and other hash functions. In Ronald Cramer, editor, *EUROCRYPT 2005*, volume 3494 of *LNCS*, pages 19–35, Aarhus, Denmark, May 22–26, 2005. Springer, Berlin, Germany.

A Extending the domain

Since a Σ-hash has two inputs, there is a natural way to regard it as a compression function and then run it in MD mode to get a full-fledged hash function. Namely, regard the first input of H_{pk} : $\mathrm{ChSet}(pk) \times \mathrm{RpSet}(pk) \to \mathrm{CmSet}(pk)$ as the data and the second as the chaining variable. For this to work however, one must be able to view the output as a chaining value, meaning we need $\mathrm{CmSet}(pk) \subseteq \mathrm{RpSet}(pk)$. Sometimes this is true, as for $\mathcal{H}$-$\mathcal{GQ}$, which in this way lends itself easily and naturally to MD iteration. But in the case of $\mathcal{SFS}$ and $\mathcal{SMS}$ we have $\mathrm{CmSet}((N, l, \mathbf{u})) = \mathbb{Z}_N^* \subsetneq \mathbb{Z}_N^+ = \mathrm{RpSet}((N, l, \mathbf{u}))$. In the case of $\mathcal{H}$-$\mathcal{Sch}$, we would like to work over on elliptic curve group because then the group size can be smaller (about 2^{160}) and computational costs are reduced. However, when

$\mathrm{CmSet}(\langle G\rangle, p, g, X) = G$ is an elliptic curve group, a group element is represented as a pair (x, y) where $x \in \mathbb{Z}_p$ and y is a bit, and this G isn't a subset of $\mathrm{RpSet}((\langle G\rangle, p, g, X)) = \mathbb{Z}_p$. However, we now present a simple and general way to get around these problems and in particular make $\mathcal{H}$-$\mathcal{SFS}$, $\mathcal{H}$-$\mathcal{SMS}$ and $\mathcal{H}$-$\mathcal{Sch}$ amenable to MD iteration. Let $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{ChSet}, \mathrm{CmSet}, \mathrm{RpSet})$ be a Σ-hash family. Let $d(\cdot)$ be an integer valued function called the data length. We now find an *embedding* e. By this we mean that $e_{pk} : \{0,1\}^{d(k)} \times \mathrm{CmSet}(pk) \to \mathrm{ChSet}(pk) \times \mathrm{RpSet}(pk)$ is an injective map for every $pk \in \left[K(1^k)\right]$ and every k. Now define $\mathcal{H}^d = (\mathsf{K}, \mathsf{H}^d, \{0,1\}^{d(\cdot)}, \mathrm{CmSet}, \mathrm{CmSet})$ by

$$\mathsf{H}^d_{pk}(m, w) = \mathsf{H}_{pk}(e^{(1)}_{pk}(m, w), e^{(2)}_{pk}(m, w))$$

where $e^{(i)}_{pk}(m, w)$ is the i-th component of the tuple $e_{pk}(m, w)$ for $i = 1, 2$. Then $\mathcal{H}^d$ is MD-compatible because the range of H^d_{pk} is the domain of its second argument and thus the second argument can be used as a chaining variable. On the other hand it is easy to see that the injectivity of e_{pk} implies that $\mathcal{H}^d$ inherits the collision-resistance of $\mathcal{H}$. So MD-iteration of $\mathcal{H}^d$ yields a full-fledged hash function which is collision-resistant.

Let us now apply this to $\mathcal{H}$-$\mathcal{SFS}$, $\mathcal{H}$-$\mathcal{SMS}$, and $\mathcal{H}$-$\mathcal{Sch}$ by finding suitable embeddings. To maximize data throughput, we should choose d as large as possible.

Suppose $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{ChSet}, \mathrm{RpSet}, \mathrm{CmSet})$ is $\mathcal{H}$-$\mathcal{SFS}$ or $\mathcal{H}$-$\mathcal{SMS}$, so that $\mathrm{ChSet}(pk) = \{0,1\}^l$ and $\mathrm{RpSet}(pk) = \mathbb{Z}^+_N$ and $\mathrm{CmSet}(pk) = \mathbb{Z}^*_N$ where $pk = (N, l, \mathbf{u})$ and $l = L(k)$, with $L(\cdot)$ being the challenge length. Let $d(\cdot) = L(\cdot) - 1$. For every N the map $f_N : \mathbb{Z}^*_N \to \{0,1\} \times \mathbb{Z}^+_N$ defined by $f_N(w) = (0, w)$ if $w \le N/2$ and $(1, N - w)$ otherwise is a bijection. Let $e_{pk} : \{0,1\}^{l-1} \times \mathbb{Z}^*_N \to \{0,1\}^l \times \mathbb{Z}^+_N$ be defined by $e_{pk}(m, w) = (m \| f^{(1)}_N(w), f^{(2)}_N(w))$ where $f^{(i)}_N(w)$ is the i-th component of the pair $f_N(w)$ for $i = 1, 2$. Then e is an embedding, so from the above $\mathcal{H}^d$ is MD-compatible and collision-resistant as long as $\mathcal{H}$ is collision-resistant. MD iterate $\mathcal{H}^d$ to get a full-fledged hash function.