Hash Functions from Sigma Protocols and Improvements to VSH

Mihir Bellare and Todor Ristov

Department of Computer Science and Engineering, University of California San Diego, 9500 Gilman Drive, La Jolla, CA 92093-0404, USA
www-cse.ucsd.edu/users/mihir, www-cse.ucsd.edu/users/tristov

Abstract. We present a general way to get a provably collision-resistant hash function from any (suitable) Σ-protocol. This enables us to both get new designs and to unify and improve previous work. In the first category, we obtain, via a modified version of the Fiat-Shamir protocol, the fastest known hash function that is provably collision-resistant based on the *standard* factoring assumption. In the second category, we provide a modified version VSH* of VSH which is faster when hashing short messages. (Most Internet packets are short.) We also show that Σ-hash functions are chameleon, thereby obtaining several new and efficient chameleon hash functions with applications to on-line/off-line signing, chameleon signatures and designated-verifier signatures.

1 Introduction

The failure of popular hash functions MD5 and SHA-1 [42, 43] lends an impetus to the search for new ones. The contention of our paper is that there will be a "niche" market for proven-secure even if not-so-fast hash functions. Towards this we provide a general paradigm that yields hash functions provably secure under number-theoretic assumptions, and also unifies, clarifies and improves previous constructs. Our hash functions have extra features such as being chameleon [25]. 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 much as PKCS#1 [24]. 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

J. Pieprzyk (Ed.): ASIACRYPT 2008, LNCS 5350, pp. 125–142, 2008.

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 [37], Fiat-Shamir [17] and GQ [20] protocols, but there are many others as well [8, 21, 29, 31, 32, 33, 36]. Briefly, our hash function is defined using the simulator underlying a strong form of the usual honest-verifier zero-knowledge property of Σ-protocols. (We stress that the computation of the hash is deterministic even though the simulator is randomized!) The collision-resistance stems from strong special soundness [7], 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 [16] and Cramer, Damgard and Mckenzie [13] 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 [30] and the latter probably not [39]. Perhaps as a consequence, our construction requires slightly stronger properties from the Σ-protocols than do the constructions of [13, 16].

SPECIFIC DESIGNS. The Schnorr [37] and GQ [20] schemes are easily shown to meet our conditions, yielding collision resistant Σ-hash functions $\mathcal{H}$-$\mathcal{S}ch$ and $\mathcal{H}$-$\mathcal{GQ}$ based, respectively, on discrete log and RSA. More interesting is the Fiat-Shamir protocol $\mathcal{FS}$ [17]. It doesn't satisfy strong special soundness 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 [29] we obtain a Σ-hash function $\mathcal{H}$-$\mathcal{SMS}$ with security based on the SRPP (Square Roots of Prime Products) assumption of [29]. We also obtain a Σ-hash $\mathcal{H}$-$\mathcal{O}k\mathcal{a}$ from Okamoto's protocol [32] and a pairing-based Σ-hash $\mathcal{H}$-$\mathcal{HS}$ from an identification protocol of [3] derived from the identity-based signature scheme of Hess [21].

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 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

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

Fig. 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.

factoring-based instantiation known of Damgård's claw free permutation-based hash function [14, 19, 25]. $\mathcal{H}$-$\mathcal{ST}$ is the hash function of Shamir and Tauman [38]. 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 $\mathcal{H}$-$\mathcal{D}\mathcal{a}$ and 9 times as fast as $\mathcal{H}$-$\mathcal{ST}$. But $\mathcal{H}$-$\mathcal{SFS}$ is amenable to pre-computation based speedup and $\mathcal{H}$-$\mathcal{D}\mathcal{a}$ 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 4 we recall $\mathcal{H}$-$\mathcal{D}\mathcal{a}$ and $\mathcal{H}$-$\mathcal{ST}$ and justify the numbers in Figure 1. We also discuss implementation results.

Features of Σ-hash functions. Krawczyk and Rabin [25] introduced chameleon hashing. The functions they show have this property are that of [10] —$\mathcal{H}$-$\mathcal{S}\mathcal{c}\mathcal{h}$ in our taxonomy— and $\mathcal{H}$-$\mathcal{D}\mathcal{a}$. Shamir and Tauman [38] add one more example, namely $\mathcal{H}$-$\mathcal{ST}$. We add five more examples, namely $\mathcal{H}$-$\mathcal{GQ}$, $\mathcal{H}$-$\mathcal{SFS}$, $\mathcal{H}$-$\mathcal{SMS}$, $\mathcal{H}$-$\mathcal{O}\mathcal{k}\mathcal{a}$, and $\mathcal{H}$-$\mathcal{HS}$. We obtain this as a consequence of a general result (Theorem 2) showing that any Σ-hash is chameleon.

Chameleon hashing has numerous applications. One of these is Shamir and Tauman's [38] chameleon hash based method for on-line, off-line signing. 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.) This compensates to some extent for the reduced efficiency of Σ-hash functions compared to conventional ones. (MD5 and SHA-1 are not chameleon and do not allow one to use the Shamir-Tauman construction.) Another application is chameleon signatures [25], 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 [25] this is an important tool for privacy-respecting authenticity in the signing of contracts and agreements. Finally, chameleon hash functions are used in designated-verifier signatures to achieve privacy [23, 40]. 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 these applications.

Another attribute of Σ-hash functions is that they 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 [5], 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.

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

A REVERSE CONNECTION. As indicated above, Theorem 2 shows that Σ-hash functions are chameleon. Theorem 1 shows that the converse is true as well. Namely, all chameleon-hash functions are Σ-hash functions. 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 we consider theoretically interesting. We also obtain numerous new Σ-protocols, and thus identification protocols and, via [13, 16], commitment schemes, from existing chameleon hash functions such as $\mathcal{H}$-$\mathcal{Da}$ [14] and $\mathcal{H}$-$\mathcal{ST}$ [38]. 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 [10], and $\mathcal{H}$-$\mathcal{Oka}$ an extension thereof [10]. (Our other hash functions $\mathcal{H}$-$\mathcal{GQ}$, $\mathcal{H}$-$\mathcal{SFS}$, $\mathcal{H}$-$\mathcal{SMS}$ and $\mathcal{H}$-$\mathcal{HS}$ are new.) The re-derivation of these two hash functions as Σ-hashes sheds new light on the designs and shows how the Σ paradigm explains and unifies previous constructs.

But the most interesting connection in this regard is one we make between VSH [11] and $\mathcal{H}$-$\mathcal{SMS}$, the Σ-hash function emanating from the protocol of Micali and Shamir [29]. 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 [11] 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 [11] with a proof based on the VSSR assumption of [11]. Our contribution is a modification VSH* of VSH that is faster 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.

We provide analogous improvements for the Fast-VSH variant of VSH provided by [11]. 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 [1, 4, 12] 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.

RELATED WORK. Damgård [14] presents a construction of collision-resistant hash functions from claw-free permutation pairs [19]. As noted above, his factoring-based instantiation, based on [19] and also considered in [25, 38], is slower than our $\mathcal{H}$-$\mathcal{SFS}$.

Ishai, Kushilevitz and Ostrovsky [22] 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 [34] under DCRA) are also both slower than known ones and based on stronger assumptions.

Charles, Goren and Lauter [9] present a construct based on the assumed hardness of some problems related to elliptic curves. Their constructs are slower than ours and additionally are based on assumptions that are non-standard and should be treated with care [41]. Lyubashevsky, Micciancio, Peikert and Rosen [27] present a fast hash function SWIFFT with an asymptotic security proof based on assumptions about the hardness of lattice problems [26, 35], but the proof would not seem to yield guarantees for the parameter sizes proposed in [27]. In contrast, our reductions are tight and the proofs provide guaranties for standard values of the security parameters. Bellare and Micciancio's construction [2] (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.

2 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 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 $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \mathrm{CmSet}, \mathrm{ChSet}, \mathrm{RpSet})$ [7] 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.

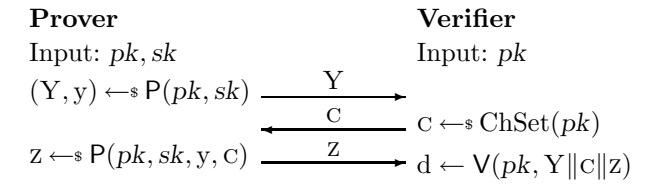

Fig. 2. Σ-protocol. Keys pk and sk are produced using key-generation algorithm K.

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 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. 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 [7]. Strong HVZK seems to be new, but is natural since we will find many protocols that posses it.

Collision-resistant hash functions. A family of n-input hash functions (where $n \geq 1$ is a constant) is a tuple $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{D}_1, \ldots, \mathrm{D}_n, \mathrm{R})$. The key-generation algorithm KG takes input 1^k and returns a key K describing a particular function $\mathsf{H}_K : \mathrm{D}_1(K) \times \ldots \mathrm{D}_n(K) \to \mathrm{R}(K)$. As this indicates, $\mathrm{D}_1, \ldots, \mathrm{D}_n, \mathrm{R}$ 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.

3 Σ-Hash Theory

This section covers the theory of Σ-hash functions. We present and justify the $\Sigma 2\mathsf{H}$ transform that turns a Σ-protocol $\mathcal{SP} \in \Sigma(\text{sss}) \cap \Sigma(\text{StHVZK})$ 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. Finally we relate Σ and chameleon hash functions. In Section 4 we discuss the practical and performance aspects of our Σ-hash functions.

The transform. We show how to build a collision-resistant hash function from any Σ-protocol $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \text{CmSet}, \text{ChSet}, \text{RpSet}) \in \Sigma(\text{sss}) \cap \Sigma(\text{StHVZK})$ that satisfies strong special soundness and strong HVZK. Let StSim be a strong HVZK simulator for $\mathcal{SP}$. We define the 2-input family of hash functions $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \text{ChSet}, \text{CmSet}, \text{RpSet})$ by $\mathsf{KG} = \mathsf{K}^{(1)}$ and $\mathsf{H}_{pk}(\text{C}, \text{Z}) = \mathsf{StSim}(pk, \text{C}, \text{Z})$, where $\mathsf{K}^{(1)}$ is the algorithm that on input 1^k lets $(pk, sk) \leftarrow\!\!\$\ \mathsf{K}(1^k)$ and returns pk. 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 $\mathcal{H} = \Sigma 2\mathsf{H}(\mathcal{SP})$ 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 1. *Let $\mathcal{SP} = (\mathsf{K}, \mathsf{P}, \mathsf{V}, \text{CmSet}, \text{ChSet}, \text{RpSet}) \in \Sigma(sss) \cap \Sigma(\text{StHVZK})$ be a Σ-protocol. Let $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \text{ChSet}, \text{RpSet}, \text{CmSet}) = \Sigma 2\mathsf{H}(\mathcal{SP})$ 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 $\mathbf{Adv}^{\text{cr}}_{H,B}(k) \leq \mathbf{Adv}^{\text{sss-na}}_{\mathcal{SP},A}(k)$, and the running time of B is that of A.* ∎

The proof of this theorem, given in [6], 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. To construct Σ-hash functions we now seek Σ-protocols which we can show are in $\Sigma(\text{sss}) \cap \Sigma(\text{StHVZK})$.

Overview of constructions. We begin, as illustrative examples, with the Schnorr [37] and GQ [20] Σ-protocols, which we can easily show to have the desired properties. The discrete log based Σ-hash $\mathcal{H}$-$\mathcal{Sch}$ obtained in the first case is that of [10] and its re-derivation as a Σ-hash sheds new light on its design and also shows how the Σ-hash paradigm unifies and explains existing work. The RSA based Σ-hash $\mathcal{H}$-$\mathcal{GQ}$ obtained in the second case is new. More interesting is the Fiat-Shamir [17] Σ-protocol. It doesn't satisfy strong special soundness, but we modify it to a Σ-protocol $\mathcal{SFS}$ that we prove is in $\Sigma(\text{sss}) \cap \Sigma(\text{StHVZK})$

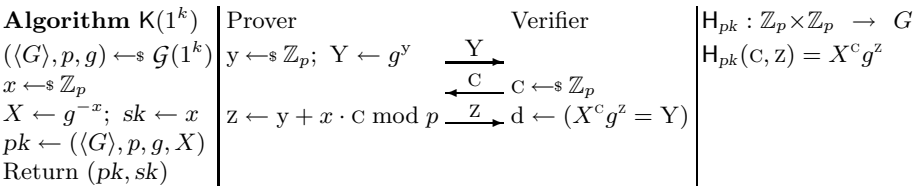

Fig. 3. $\mathcal{Sch}$ Σ-protocol and the derived Σ-hash family, where $\mathcal{G}$ is a prime-order group generator

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 [29]. In [6] we show how to get another discrete-log based Σ-hash from Okamoto's protocol [32] and a pairing based one from the $\mathcal{HS}$ protocol [3, 21]. We proceed to the details.

$\mathcal{Sch}$. 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 [37] 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 $\mathcal{Sch} \in \Sigma(\mathrm{StHVZK})$ 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 $\mathrm{C}_1 \neq \mathrm{C}_2$. To sss-adversary A, this leads us to associate 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 1 $\mathcal{H}$-$\mathcal{Sch}$ is collision-resistant under the same assumption.

$\mathcal{GQ}$. We fix a prime-exponent RSA generator with associated challenge length $L(\cdot)$, by which we mean a PT algorithm $\mathcal{G}_{\mathrm{rsa}}$ 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 [20] 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 $\mathrm{Y} = \mathrm{Z}^{\mathrm{C}} \mathrm{Z}^2 \bmod N$, 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

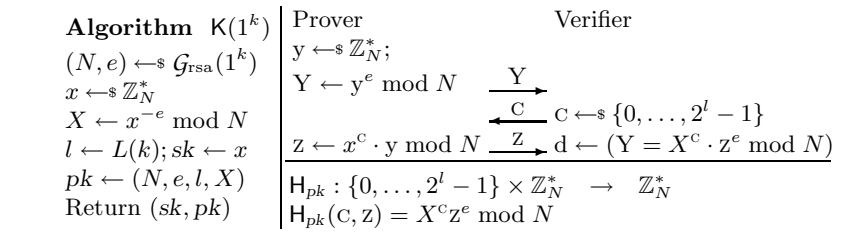

Fig. 4. $\mathcal{GQ}$ Σ-protocol and the derived Σ-hash family, where $\mathcal{G}_{\rm rsa}$ is a prime exponent RSA generator with associated challenge length L

found via the extended gcd algorithm. (This is where we use the fact that e is prime.) Then for all k we have $\mathbf{Adv}^{\rm sss}_{\mathcal{GQ},A}(k) \leq \mathbf{Adv}^{\rm rsa}_{\mathcal{G}_{\rm rsa},I}(k)$, 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 1, $\mathcal{H}$-$\mathcal{GQ}$ is collision-resistant under the same assumption.

$\mathcal{FS}$ 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 $1 \leq i \leq l$ 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 $w \leq N/2$ 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 CmSet, ChSet are 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. In [6] we show how to associate to any PT sss-adversary A a PT factoring adversary B such that for all $k \in \mathbb{N}$ we have $\mathbf{Adv}^{\rm sss}_{\mathcal{SFS},A} \leq 2 \cdot \mathbf{Adv}^{\rm fac}_{\mathcal{G}_{\rm mod},B}(k)$, where the latter is defined 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 $r \leftarrow\!\!\$\ B(N)$. (Briefly, if $\mathrm{Y} \| \mathrm{C}_1 \| \mathrm{Z}_1$ and $\mathrm{Y} \| \mathrm{C}_2 \| \mathrm{Z}_2$ are accepting transcripts then if $\mathrm{C}_1 \neq \mathrm{C}_2$ we obtain the square root of some component of the public key and if $\mathrm{C}_1 = \mathrm{C}_2$ but $\mathrm{Z}_1 \neq \mathrm{Z}_2$ then $\mathrm{Z}_1, \mathrm{Z}_2$ are non-trivial square roots of the same square and we can factor N.) This shows that $\mathcal{SFS}$ has strong special soundness under the *standard* hardness of factoring assumption. Now, the algorithm that on input $pk = (N, l, \mathbf{u})$ lets $\mathrm{C} \leftarrow\!\!\$\ \{0,1\}^l$; $\mathrm{Z} \leftarrow\!\!\$\ Z_N^+$; $\mathrm{Y} \leftarrow \mathbf{u}^{\mathrm{C}} \cdot \mathrm{Z}^2 \bmod N$ 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. Theorem 1 implies that $\mathcal{H}$-$\mathcal{SFS}$ is collision resistant under the standard factoring assumption.

$\mathcal{MS}$ AND $\mathcal{SMS}$. The Micali-Shamir protocol [29] is a variant of $\mathcal{FS}$ in which verification time is reduced by choosing the coordinates of $\mathbf{u}$ to be small primes.

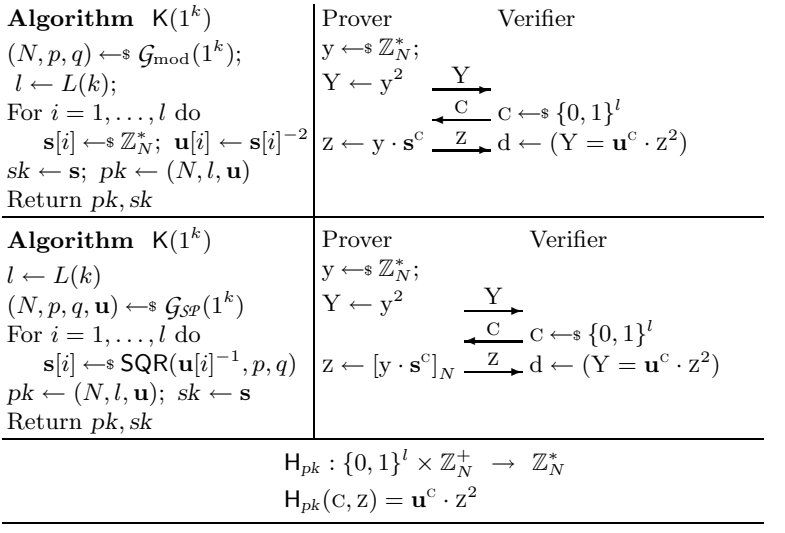

Fig. 5. $\mathcal{FS}$, $\mathcal{SFS}$, $\mathcal{MS}$ and $\mathcal{SMS}$ protocols and the Σ-hash derived from $\mathcal{SFS}$, $\mathcal{SMS}$. The upper left key-generation algorithm is that of $\mathcal{FS}$ and $\mathcal{SFS}$, while the lower left one is that of $\mathcal{MS}$ and $\mathcal{SMS}$.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

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

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

The SRPP (Square Root of Prime Products) assumption [29] 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 $\Sigma 2\mathsf{H}(\mathrm{StHVZK})$ 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 [6] under the SRPP assumption. Theorem 1 now implies that $\mathcal{H}$-$\mathcal{SMS}$ is collision-resistant under the SRPP assumption.

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. This is captured by the following.

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

Refer to [6] for the proof of the above and the relevant definitions. As a consequence, we obtain the following new chameleon hash functions: $\mathcal{H}\text{-}\mathcal{GQ}$, $\mathcal{H}\text{-}\mathcal{SFS}$, $\mathcal{H}\text{-}\mathcal{SMS}$, $\mathcal{H}\text{-}\mathcal{Oka}$, $\mathcal{H}\text{-}\mathcal{HS}$. ($\mathcal{H}\text{-}\mathcal{Sch}$ was already known to be chameleon [25].) This yields numerous new and more efficient instantiations of on-line/off-line signatures [38], chameleon signatures [25] and designated-verifier signatures [23, 40].

Even more interestingly, we prove the converse. The following theorem says that any chameleon hash family is a Σ-hash family, meaning the result of applying our $\Sigma 2\mathsf{H}$ transform to some Σ-protocol.

Theorem 3. *Let* $\mathcal{H} = (\mathsf{KG}, \mathsf{H}, \mathrm{ChSet}, \mathrm{CmSet}, \mathrm{RpSet})$ *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* $\mathcal{SP}$.

The proof is in [6]. Applying this to known chameleon-hash functions like $\mathcal{H}\text{-}\mathcal{Da}$ [14, 25] and $\mathcal{H}\text{-}\mathcal{ST}$ [38] yields new Σ-protocols and hence new identification schemes and, via [13, 16], new commitment schemes.

4 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.

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 [15, 28]. 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 $\mathcal{H}\text{-}\mathcal{GQ}$, 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 [6] 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,

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 precomputation and the space is the number of group elements that need to be stored.

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

$$\mathsf{H}_{pk}(\mathrm{C}, \mathrm{Z}) = \mathbf{u}^{\mathrm{C}} \cdot \mathrm{Z}^2 \cdot v^{f_N(\mathrm{Z})} \bmod N, \tag{1}$$

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 $\Sigma 2\mathsf{H}$ 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.

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 [25, 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.

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 (1). (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, 3.2 GHz machine, running Linux 2.6 kernel and using the gmp library [18].

Comparisons. We now assess performance of previous schemes, justifying claims in Section 1. Damgård [14] shows how to construct collision-resistant hash functions from claw-free permutations [19]. Of various factoring-based instantiations of his construction, the one of [19, 25], 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 [38] 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.

5 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 [11] 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 [11] 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 (1)— 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 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

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

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

this is not the only possible attack, but the proof of strong special soundness of $\mathcal{SMS}$ [6] 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 Dual Pentium IV, 3.2 GHz machine running Linux kernel 2.6 and using the gmp library [18]. 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 [11]) 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 are tested on the same platform.

References

1. Andreeva, E., Neven, G., Preneel, B., Shrimpton, T.: Seven-property-preserving iterated hashing: ROX. In: Kurosawa, K. (ed.) ASIACRYPT 2007. LNCS, vol. 4833, pp. 130–146. Springer, Heidelberg (2007)
2. Bellare, M., Micciancio, D.: A new paradigm for collision-free hashing: Incrementality at reduced cost. In: Fumy, W. (ed.) EUROCRYPT 1997. LNCS, vol. 1233, pp. 163–192. Springer, Heidelberg (1997)
3. Bellare, M., Namprempre, C., Neven, G.: Security proofs for identity-based identification and signature schemes. In: Cachin, C., Camenisch, J.L. (eds.) EUROCRYPT 2004. LNCS, vol. 3027, pp. 268–286. Springer, Heidelberg (2004)
4. Bellare, M., Ristenpart, T.: Multi-property-preserving hash domain extension and the EMD transform. In: Lai, X., Chen, K. (eds.) ASIACRYPT 2006. LNCS, vol. 4284, pp. 299–314. Springer, Heidelberg (2006)
5. Bellare, M., Ristenpart, T.: Hash functions in the dedicated-key setting: Design choices and MPP transforms. In: Arge, L., Cachin, C., Jurdziński, T., Tarlecki, A. (eds.) ICALP 2007. LNCS, vol. 4596, pp. 399–410. Springer, Heidelberg (2007)
6. Bellare, M., Ristov, T.: Hash Functions from Sigma Protocols and Improvements to VSH. Full Version of this paper. IACR eprint archive (2008)
7. Bellare, M., Shoup, S.: Two-tier signatures, strongly unforgeable signatures, and Fiat-Shamir without random oracles. In: Okamoto, T., Wang, X. (eds.) PKC 2007. LNCS, vol. 4450, pp. 201–216. Springer, Heidelberg (2007)

8. Beth, T.: Efficient zero-knowledge identification scheme for smart cards. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330, pp. 77–84. Springer, Heidelberg (1988)
9. Charles, D., Goren, E., Lauter, K.: Cryptographic hash functions from expander graphs. In: Second NIST Hash Function Workshop (2006)
10. Chaum, D., Heijst, E.V., Pfitzmann, B.: Cryptographically strong undeniable signatures, unconditionally secure for the signer. In: Feigenbaum, J. (ed.) CRYPTO 1991. LNCS, vol. 576, pp. 470–484. Springer, Heidelberg (1992)
11. Contini, S., Lenstra, A.K., Steinfeld, R.: VSH, an efficient and provable collision-resistant hash function. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 165–182. Springer, Heidelberg (2006)
12. Coron, J.-S., Dodis, Y., Malinaud, C., Puniya, P.: Merkle-Damgård revisited: How to construct a hash function. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 430–448. Springer, Heidelberg (2005)
13. Cramer, R., Damgrd, I., MacKenzie, P.: Efficient zero-knowledge proofs of knowledge without intractability assumptions. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274. Springer, Heidelberg (2002)
14. Damgård, I.: Collision free hash functions and public key signature schemes. In: Günther, C.G. (ed.) EUROCRYPT 1988. LNCS, vol. 330. Springer, Heidelberg (1988)
15. Damgård, I.: A design principle for hash functions. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 416–427. Springer, Heidelberg (1990)
16. Damgård, I.: On the existence of bit commitment schemes and zero-knowledge proofs. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 17–27. Springer, Heidelberg (1990)
17. Fiat, A., Shamir, A.: How to prove yourself: Practical solutions to identification and signature problems. In: Odlyzko, A.M. (ed.) CRYPTO 1986. LNCS, vol. 263, pp. 186–194. Springer, Heidelberg (1987)
18. The GNU MP bignum library, `http://gmplib.org/`
19. Goldwasser, S., Micali, S., Rivest, R.: A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks. SIAM J. on Computing 17 (1988)
20. Guillou, L.C., Quisquater, J.-J.: A "paradoxical" indentity-based signature scheme resulting from zero-knowledge. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 216–231. Springer, Heidelberg (1990)
21. Hess, F.: Efficient identity based signature schemes based on pairings. In: Nyberg, K., Heys, H.M. (eds.) SAC 2002. LNCS, vol. 2595, pp. 310–324. Springer, Heidelberg (2003)
22. Ishai, Y., Kushilevitz, E., Ostrovsky, R.: Sufficient conditions for collision-resistant hashing. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 445–456. Springer, Heidelberg (2005)
23. Jakobsson, M., Sako, K., Impagliazzo, R.: Designated verifier proofs and their applications. In: Maurer, U.M. (ed.) EUROCRYPT 1996. LNCS, vol. 1070, pp. 143–154. Springer, Heidelberg (1996)
24. Jonsson, J., Kaliski, B.: Public-key cryptography standards (PKCS) #1: RSA cryptography, specifications version 2.1. Internet RFC 3447 (2003)
25. Krawczyk, H., Rabin, T.: Chameleon hashing and signatures. In: ISOC Network and Distributed System Security Symposium (NDSS 2000) (2000)
26. Lyubashevsky, V., Micciancio, D.: Generalized compact knapsacks are collision resistant. In: Bugliesi, M., Preneel, B., Sassone, V., Wegener, I. (eds.) ICALP 2006. LNCS, vol. 4052, pp. 144–155. Springer, Heidelberg (2006)

27. Lyubashevsky, V., Micciancio, D., Peikert, C., Rosen, A.: SWIFFT: a modest proposal for FFT hashing. In: Nyberg, K. (ed.) FSE 2008. LNCS, vol. 5086, pp. 54–72. Springer, Heidelberg (2008)
28. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
29. Micali, S., Shamir, A.: An improvement of the Fiat-Shamir identification and signature scheme. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 244–247. Springer, Heidelberg (1990)
30. Naor, M.: Bit commitment using pseudorandomness. Journal of Cryptology 4(2), 151–158 (1991)
31. Ohta, K., Okamoto, T.: A modification of the Fiat-Shamir scheme. In: Goldwasser, S. (ed.) CRYPTO 1988. LNCS, vol. 403, pp. 232–243. Springer, Heidelberg (1990)
32. Okamoto, T.: Provably secure and practical identification schemes and corresponding signature schemes. In: Stinson, D.R. (ed.) CRYPTO 1993. LNCS, vol. 773. Springer, Heidelberg (1994)
33. Ong, H., Schnorr, C.-P.: Fast signature generation with a Fiat-Shamir like scheme. In: Damgård, I.B. (ed.) EUROCRYPT 1990. LNCS, vol. 473, pp. 432–440. Springer, Heidelberg (1991)
34. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, p. 223. Springer, Heidelberg (1999)
35. Peikert, C., Rosen, A.: Efficient Collision-Resistant Hashing from Worst-Case Assumptions on Cyclic Lattices. In: Halevi, S., Rabin, T. (eds.) TCC 2006. LNCS, vol. 3876, pp. 145–166. Springer, Heidelberg (2006)
36. Sakai, R., Ohgishi, K., Kasahara, M.: Cryptosystems based on pairing. In: SCIS 2000 (2000)
37. Schnorr, C.-P.: Efficient signature generation by smart cards. Journal of Cryptology 4(3), 161–174 (1991)
38. Shamir, A., Tauman, Y.: Improved online/offline signature schemes. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, p. 355. Springer, Heidelberg (2001)
39. Simon, D.R.: Finding collisions on a one-way street: Can secure hash functions be based on general assumptions? In: Nyberg, K. (ed.) EUROCRYPT 1998. LNCS, vol. 1403, pp. 334–345. Springer, Heidelberg (1998)
40. Steinfeld, R., Wang, H., Pieprzyk, J.: Efficient extension of standard Schnorr/RSA signatures into universal designated-verifier signatures. In: Bao, F., Deng, R., Zhou, J. (eds.) PKC 2004. LNCS, vol. 2947, pp. 86–100. Springer, Heidelberg (2004)
41. Tillich, J.-P., Zemor, G.: Collisions for the LPS expander graph hash function. In: Smart, N.P. (ed.) EUROCRYPT 2008. LNCS, vol. 4965, pp. 254–269. Springer, Heidelberg (2008)
42. Wang, X., Yin, Y.L., Yu, H.: Finding collisions in the full SHA-1. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 17–36. Springer, Heidelberg (2005)
43. Wang, X., Yin, Y.L., Yu, H.: How to break MD5 and other hash functions. In: Cramer, R. (ed.) EUROCRYPT 2005. LNCS, vol. 3494, pp. 19–35. Springer, Heidelberg (2005)