Strengthening Digital Signatures via Randomized Hashing

Shai Halevi[*] Hugo Krawczyk[†‡]

January 30, 2007

Abstract

We propose randomized hashing as a mode of operation for cryptographic hash functions intended for use with standard digital signatures and without necessitating of any changes in the internals of the underlying hash function (e.g., the SHA family) or in the signature algorithms (e.g., RSA or DSA). The goal is to free practical digital signature schemes from their current reliance on strong collision resistance by basing the security of these schemes on significantly weaker properties of the underlying hash function, thus providing a *safety net* in case the (current or future) hash functions in use turn out to be less resilient to collision search than initially thought.

We design a specific mode of operation that takes into account engineering considerations (such as simplicity, efficiency and compatibility with existing implementations) as well as analytical soundness. Specifically, the scheme entails unmodified use of the hash function with randomization applied only to the message before it is input to the hash function. We formally show the sufficiency of an assumption significantly weaker than collision-resistance for proving the security of the scheme.

We also contribute to the standardization of a randomized hashing mode by providing a full and detailed spec that instantiates our scheme, provides the full benefits guaranteed by our results, and is ready for implementation and integration with existing applications.

1 Introduction

Recent cryptanalytical advances in the area of collision-resistant hash functions (CRHF) [9, 5, 16, 6, 17, 31, 32, 33, 34], especially the attacks against MD5 and SHA-1, have shaken our confidence in the security of existing hash functions as well as in our ability to design secure CRHF. These attacks remind us that cryptography is founded on heuristic constructions whose security may be endangered unexpectedly, and highlight the importance of designing mechanisms that require as little as possible from their basic cryptographic building blocks. In particular, they indicate that one should move away (to the extent possible) from schemes whose security fundamentally depends on full collision resistance.

The most prominent example of schemes that are endangered by these attacks are digital signatures where collisions in the hash function directly translate into forgeries. The goal of this work is

[*]IBM T.J. Watson Research Center, Yorktown Heights, New York 10598. Email: shaih@alum.mit.edu.

[†]IBM T.J. Watson Research Center, Yorktown Heights, New York 10598. Email: hugo@ee.technion.ac.il.

[‡]A short version of this paper appears in Crypto'06. See `http://www.ee.technion.ac.il/~hugo/rhash/` for updates and for more information on the proposed scheme, including specification and implementation.

to free existing (standardized) signature schemes from their dependence on full collision-resistance of the underlying hash function, while making as small as possible modifications to the signature and hashing algorithms. Specifically, we propose a *randomized mode of operation* for hash functions such that (i) no change to the underlying hash functions (e.g., the SHA family), the signature algorithms (e.g., RSA, DSA), or their implementations is required; (ii) changes to the signing process are confined to the minimum required by any randomization scheme, namely, the choice of a short random string by the signer and transporting this string to the receiver; and (iii) *security of the resultant signature scheme does not depend on the resistance of the hash function to off-line collision attacks.*

Armoring signature schemes with this mode of operation provides a *safety net* for the security of digital signatures in the case that the collision resistance of the underlying hash function is broken or weakened. At the same time, by following important engineering considerations, as in points (i) and (ii) above, one facilitates the adoption of the resultant schemes into practice. In particular, treating the hash function as a black box allows to preserve existing implementations of these functions or their future replacements. Moreover, the many other applications of hash functions (e.g., MAC, PRFs, commitments, fingerprinting, etc.) need not be changed or adapted to a new hash design; instead, these applications may use the hash function, or any "drop-in replacement", exactly as they do now.

Collision resistance and the Merkle-Damgård (M-D) construction. Roughly, a (length-decreasing) function H is collision resistant if given the description of H, no feasible attacker can find two messages M, M' such that $H(M) = H(M')$, except with insignificant probability.[1] Contemporary constructions of (allegedly) collision-resistant hash functions follow the so called Merkle-Damgård (M-D) iterated construction [18, 10]. Such constructions start with a *compression function* h that maps input pairs (c, m) into an output c' where c and c' are of fixed length n (e.g., $n = 160$) and m is of fixed length b (e.g., $b = 512$). Given such a compression function h and a fixed n-bit initial value (IV), denoted c_0, a hash function H on arbitrary-length inputs is defined by iterating h as follows: On input message M, the message is broken into b-bit blocks $M = (m_1, m_2, \ldots, m_L)$, then one computes $c_i \leftarrow h(c_{i-1}, m_i)$ for $i = 1, 2, \ldots, L$ and finally the last n-bit value c_L is output as the result of $H(M)$.

Throughout this paper we typically denote the (iterated) hash function by H and its compression function by h. For an M-D function H, we also sometimes write $H^{c_0}(M)$ when we want to explicitly name the initial value c_0. To handle non-suffix-free message spaces, the M-D construction appends the length at the end of the input and uses some padding rules. We will initially ignore these issues and assume that the input consists of an integral number of blocks and that it is taken from a suffix-free set; we will return to the issue of length appending in Section 5.

Target Collision Resistance. Signature schemes that do not depend on full collision resistance were constructed in the influential work of Naor and Yung [21] who introduced the notion of *universal one-way hash functions, or UOWHF*. Later, Bellare and Rogaway [3] renamed them to the more descriptive (and catchy) name of *target collision resistant* (TCR) hash functions, a term that we adopt here. Roughly, a family of hash functions $\{H_r\}_{r \in R}$ (for some set R) is target collision resistant if no efficient attacker A can win the following game, except with insignificant probability: A chooses a first message M, then receives a random value $r \in_{\mathrm{R}} R$, and it needs to find a second

[1]Formalizing collision resistance requires to consider H as a family of functions rather than as a single function; we ignore this technicality for the informal discussion here.

message $M' \neq M$ such that $H_r(M') = H_r(M)$. The value r is called a hashing key, or a *salt*. In [3], TCR families were used to extend signature schemes on short messages into signature schemes on arbitrary messages: To sign a long message M, the signer chooses a fresh random salt r and then uses the underlying signature scheme to sign the pair $(r, H_r(M))$.

Enhanced target collision resistance. Standard signature schemes such as RSA and DSA are defined using a deterministic hash function H and designed to only sign the hash value $H(M)$. When moving to a TCR-based scheme one replaces $H(M)$ with $H_r(M)$ but also needs to sign the salt r. Unfortunately, certain signature schemes, such as DSA, do not accommodate the signing of r in addition to $H_r(M)$. Even in cases, such as RSA, where both r and $H_r(M)$ can be included under the signed block (RSA moduli are long enough to accommodate both values) signing r requires changing the message encoding standards (such as PKCS#1) which may be impractical in some settings. To solve these difficulties, we propose a strengthened mode of operation, that provides a stronger security guarantee than TCR. The new notion, that we call *enhanced TCR* (eTCR), is sufficiently strong to ensure security of the resultant signatures even if we only apply the underlying signature to $H_r(M)$ and *do not sign the salt* r. Specifically, the TCR attack scenario is modified so that after committing to M and receiving the salt r, the attacker can supply a second message M' *and a second salt* r', and it is considered successful if $(r, M) \neq (r', M')$ but $H_r(M) = H_{r'}(M')$.

1.1 Our Randomization Schemes

The main contribution of this work is in presenting simple and efficient randomization schemes that use any Merkle-Damgård iterated hash function as-is (i.e., in a black-box fashion) and have short salts, and which we prove to be TCR and/or eTCR under weak assumptions on the underlying compression function. Specifically, we relate the security of our schemes to "second-preimage resistance" properties of the compression function. Recall that a compression function h is called second preimage resistant (SPR) if given a uniformly random pair (c, m) it is infeasible to find a second pair $(c', m') \neq (c, m)$ such that $h(c', m') = h(c, m)$.

The first scheme that we consider, denoted H_r, simply XOR's a random b-bit block r to each message block before inputting it into the hash function. That is,

$$H_r^c(m_1, \ldots, m_L) \stackrel{\text{def}}{=} H^c(m_1 \oplus r, \ldots, m_L \oplus r). \tag{1}$$

We show this scheme to be TCR under SPR-like assumption on the underlying compression function h (see below). On the other hand, the H_r construction is clearly not eTCR; in order to obtain an eTCR scheme we modify H_r by prepending the salt block r to the message (in addition to xor-ing r to every message block as before), thus computing

$$\tilde{H}_r^c(M) \stackrel{\text{def}}{=} H_r^c(0|M) = H^c(r, m_1 \oplus r, \ldots, m_L \oplus r) \tag{2}$$

Remarkably, for any M-D iterated hash function H we can prove that $\tilde{H}_r$ is enhanced-TCR under *the same relaxed conditions on the compression function* that we use to prove that H_r is TCR.

These precise conditions are presented in Section 2. Roughly, we show two properties of the compression function h that are sufficient for H_r to be TCR (resp. for $\tilde{H}_r$ to be eTCR). Both properties are related to the second-preimage resistance of h. One property, c-SPR, is rather natural but also rather strong, and is provided mostly as a toy problem for cryptanalysts to sink their teeth in. The other property, e-SPR, is less natural but is closer to SPR and reflects more accurately the "real hardness" of the constructions H_r and $\tilde{H}_r$. We also note that c-SPR is related to the "hierarchy of collision resistance" of Mironov [19], and that e-SPR is a weaker property than the L-order TCR property of Hong et al. [15].

We stress that ultimately, the question of whether or not a particular compression function (such as those in the SHA family) is e-SPR can only be addressed by cryptanalysts trying to attack this property. As we explain in Section 2, breaking the compression function in the sense of e-SPR is a much harder task than just finding collisions for it (see also remark on the necessity of e-SPR in Section 3). Moreover, the randomized setting represents a *fundamental shift* in the attack scenario relative to the traditional use of deterministic hash functions. In the latter case, the attacker can find collisions via (heavy) *off-line* computation and later use these collisions in a signature forgery attack at any time and with any signer. In contrast, in the randomized setting, a meaningful collision is one that utilizes a random, unpredictable r *chosen by the signer anew with each signature* and revealed to the attacker only with the signature on the message M (in particular, only after the attacker committed to the message M).

Can the signer cheat? A possible objection to the randomized setting is that it may allow a signer to find two messages with the same hash value. This, however, represents no violation of signature security (c.f. the standard definition of Goldwasser, Micali, and Rivest [13]). Specifically, the potential ability of the signer to find collisions does not allow any other party to forge signatures thus protecting the signer against malicious parties. At the same time, recipients of signatures are also protected since the signer cannot disavow a signature by presenting a collision; the simple rule is, as in the case of human signatures, that the signer is liable for any message carrying his valid signature (even if the signer can show a collision between two messages). It is also worth noting that our schemes have the property that as long as the underlying hash function is fully collision resistant then even the signer cannot find collisions. More generally, the signature schemes resulting from our randomization modes are never weaker (or less functional) than the schemes that use a deterministic hash function, not even when the underlying hash function is truly collision resistant; and, of course, our schemes are much more secure once the hash function turns out not to be collision resistant.

Note. We do not make any claims regarding unconventional uses of digital signatures that assume properties beyond unforgeability. This is the case, for example, in applications that use a signature as a commitment to a message or in the counter-signature implementation of [30]. In these two cases it is assumed (often implicitly) that the signature scheme is in itself collision resistant, namely, that it is infeasible (even for the signer) to find two messages with the same signature. This property does not follow from, nor it implies, unforgeability, and hence should not be assumed to be an inherent property of secure signatures (ECDSA is an example of an unforgeable scheme that does not ensure this property even if the underlying hash function is fully collision resistant).

1.2 A Practical Instantiation

As seen, our randomized hashing schemes require no change to the underlying (compression and iterated) hash function. These schemes can be used, as described before, with algorithms such as RSA and DSA, to provide for digital signatures that remain secure even in the presence of collisions for the underlying hash functions. For this, the value $H(M)$ currently signed by the standard schemes needs to be replaced with $H_r(M)$ or with $\tilde{H}_r(M)$. However, while in the case of H_r one needs to also sign the salt r, in the $\tilde{H}_r$ case the latter is not needed. In this sense,

$\tilde{H}_r$ provides for a more flexible and practical scheme and hence it is the one that we suggest for adoption into practice. In particular, using $\tilde{H}_r$, the salt r may be transported by an application as part of (or in addition to) a message, or r can be included under the signature itself (under the signed value in RSA or by re-using the random $r = g^k$ component in DSA). We further discuss these issues in Section 5.

RMX: A fully specified scheme. In Appendix D we present a fully specified message randomization scheme, named RMX, which follows the design of our eTCR $\tilde{H}_r$ scheme. Our results imply that if one uses RMX as a front-end to a typical hash-then-sign signature scheme (such as RSA or DSS) then one obtains secure signatures under significantly weaker assumptions on the underlying hash function. The RMX scheme is specified for use with Merkle-Damgård hash functions (and can be adapted to other iterative hash functions) and is intended for standardization as a mode of operation for hash functions. In [7] we report on implementation experience with RMX which indicates the practical viability and simplicity of the scheme.

1.3 Further Remarks

Related Schemes. Bellare and Rogaway [3] explored the problem of constructing TCR hashing for long messages from TCR functions that work on short inputs, and showed that the M-D iteration of a TCR compression function *does not necessarily* result in a long-message TCR. Instead, they presented a few other constructions of long-message TCR hash functions from short-message TCR compression functions. Shoup [27] continued this line of research and offered the following elegant scheme that comes close to the original M-D iterated construction. The salt in Shoup's scheme consists of one b-bit message block r and a set S of $\log L$ chaining variables $s_0, s_1, \ldots$, where L is the number of message blocks. The scheme consists of a regular M-D construction except that (i) every message block is xor-ed with r before it is input to the compression function h, and (ii) the intermediate values c_i output by each iteration of h are xor-ed with one of the s_i's to form the chaining value input into the following application of h. That is, the M-D scheme is modified as follows: for $i = 1, 2, \ldots, L$, $c_i \leftarrow h(c_{i-1} \oplus s_{j_i}, m_i \oplus r)$ where j_i chooses one of the elements in the set S. Shoup proved that if the compression function is SPR then the construction yields a TCR function on messages of arbitrary length.

Note that our H_r scheme (Eqn. (1)) is similar to Shoup's in that the input blocks are xor-ed with the random r. However, the two schemes differ substantially in that: (i) H_r uses the hash function as a black box while Shoup's scheme requires the change of the internals of H to accommodate the xor-ing of the s_i's values with the partial results of the compression function; and (ii) in H_r the amount of randomness is fixed while in Shoup's it is logarithmic in the length of the message and hence non-constant; this requires significantly more random bits and results in longer signatures. In addition, Shoup's scheme (like H_r) is not eTCR. For the latter property, we need the modified $\tilde{H}_r$ scheme (Eqn. (2)).

Why not HMAC? One interesting question is why not take as the randomization mode a well established construction such as HMAC. It is easy to see that for the purpose of TCR hashing the HMAC scheme (with the salt used as a random but known key) is not stronger than just pre-pending a random value to a message, a measure of limited effect against recent cryptanalysis. Other measures, such as appending randomness at the end, or even in the middle of a message, are even worse in that they do not help against collision attacks. In particular, this shows that the simplicity of our schemes cannot be explained by the naïve view that "any randomization of the input to the hash function will work." Yet, one analogy to the case of HMAC is illustrative here: the fact that the HMAC design (as a message authentication code) has not been endangered by the latest collision attacks is due to its deliberate non-reliance on full collision resistance. We believe that randomized hashing could have a similar significant effect on the future security of digital signatures.

Relations between various notions. In our presentation we use quite a few notions of security for hash functions, some new and others well known (see [25]). Articulating the relations between these notions is *not* the focus of this paper. In some places we comment about implication or separation between these notions, but we do not explicitly show separation examples in this extended abstract. (This will divert attention from what we see as the heart of this work, namely the analysis of the practical constructions H_r and $\tilde{H}_r$.)

Organization. In Section 2 we present and discuss the SPR-like notions that we use in proving the TCR and eTCR properties of our schemes. These proofs are presented in Section 3 and Section 4, respectively. In Section 5 we discuss some practical aspects related to the integration of our schemes with standard signature schemes. Some open problems are presented in Section 6. As an additional contribution, Appendix D contains a fully detailed specification of a concrete message randomization scheme built on the basis of our results and which we intend for standardization and implementation.

2 Variants of Second-Preimage Resistant Hashing

Our goal in this work is to analyze the randomized modes of operation defined in Equations 1 and 2, finding "as weak as possible SPR-like properties" of h that suffice to ensure that H_r is TCR and $\tilde{H}_r$ is eTCR. Below we describe several "games" representing different forms of second-preimage resistance (SPR) of the compression function h (recall that h represents a single, unkeyed, function with two arguments, c and m, and H^c represents the M-D iteration of h with IV$= c$). In all cases the goal of the attacker (that we denote by S) is to present pairs (c, m) and (c', m') such that $(c, m) \neq (c', m')$ and yet $h(c, m) = h(c', m')$. The difference between the games is how the values c, m, c', m' are determined and by whom.

The case where S chooses all four values is known as the "pseudo-collision" search problem for h and is at the basis of the Merkle-Damgård (M-D) construction of collision resistance. However, in the games below the adversary's task is significantly harder since some of these values are chosen randomly and are not under the control of S. More specifically, the games are variants of "second-preimage resistance" of the function h since *in all the value m is chosen independently at random* and c', m' are chosen by the attacker at will after seeing m. The difference is in the way c is determined: at $\underline{\text{r}}$andom in r-SPR, $\underline{\text{c}}$hosen (by the attacker) in c-SPR, and $\underline{\text{e}}$valuated (as a function of m) in e-SPR:

r-SPR: S receives random m and c, and it chooses m', c'.

c-SPR: S receives random m, and it chooses c, c', m'.

e-SPR: The game is parametrized by an IV c_0. S chooses $\ell \geq 1$ values Δ_i, $i = 1, \ldots, \ell$, each of length b bits; then S receives a random $r \in \{0,1\}^b$ and c, m are set to $m = r \oplus \Delta_\ell$ and $c = H^{c_0}(r \oplus \Delta_1, \ldots, r \oplus \Delta_{\ell-1})$. Finally S chooses c', m'.[2]

Clearly, r-SPR corresponds to the standard notion of second-preimage resistance of h, while c-SPR and e-SPR are variants that we use in the analysis of our schemes, and we elaborate on them below.

The IV. The game e-SPR can be considered in either the "uniform setting" where c_0 is chosen at random and given to S as input, or the "non-uniform setting" where c_0 is a parameter of the game and S may depend on that parameter. The discussion below applies equally to both settings.

Attack parameters. When discussing the security (or hardness) of different games we denote by t and L the attacker's resources (i.e., time and length of messages used in the attack) and by ε the probability of success by the attacker. In particular, if G denotes one of the SPR or TCR games discussed in the paper then we say that a function (either h or the family H_r) is $G(\varepsilon, L, t)$ if no attacker with resources t and L can win the game with probability better than ε. (In some of the games the parameter L is irrelevant and then omitted.)

2.1 The c-SPR Game

We remark that the game c-SPR is a significantly easier variant (for the attacker) than the r-SPR and e-SPR games, since the attacker gets to choose c, after seeing m, in addition to choosing c' and m'. For example, the c-SPR game is clearly vulnerable to generic birthday attacks, whereas r-SPR and e-SPR are not.

We observe that for a compression function $h(c, m)$, the c-SPR property is equivalent to the property that the family $h_r(m, c) = h(c, m \oplus r)$ is $CR_n(n + b, n)$ as defined by Mironov [19].[3] A family of functions $\{h_r\}_r$ belongs to $CR_\ell(n', n)$ if each h_r maps n' bits to n bits, and in addition no feasible attacker can win the following game except with insignificant probability:

1. The attacker first commits to $n' - \ell$ bits of the first message (call this $(n' - \ell)$-bit string M_1),
2. Then the attacker is given the salt r,
3. The attacker wins if it can find an ℓ-bit string M_2 and a second n'-bit message M' such that for $M = M_1|M_2$ it holds that $h_r(M) = h_r(M')$.

Now, consider the following equivalent formulation of the c-SPR game: the attacker first commits to a b-bit message m, then it gets the randomness r and it needs to find c, c' and m' such that $h(c, m \oplus r) = h(c', m' \oplus r)$. Clearly, under this formulation we see that $\{h_r\}_r$ is c-SPR if and only if $\{h_r\}_r$ belongs to the class $CR_n(n + b, n)$. Mironov proved that the existence of $CR_n(n + b, n)$ families implies the existence of collision-resistant families (from $n + 1$ to n bits). Hence c-SPR implies the *existence* of collision-resistant hashing families (in particular, this shows that Simon's [28] separation result applies to black-box constructions of c-SPR; it does *not* mean, however, that the M-D iteration of a c-SPR compression function is necessarily collision resistant). Still, for

[2]We choose to define $m = r \oplus \Delta_\ell$ for notational convenience in our proofs; however, note that the xor with Δ_ℓ does not change the fact that the value of m is determined uniformly at random (and independent of the Δ values).

[3]Here we reverse the order of the inputs to h_r in order to match Mironov's formulation.

particular instantiations of hash functions, breaking c-SPR is significantly harder, in general, than the traditional pseudo-collision search. (Also, assuming that c-SPR functions exist, it is easy to construct compression functions that are c-SPR but are not resistant to pseudo-collisions.) We believe that the c-SPR game provides a useful toy-example for cryptanalysts to develop tools for TCR-type attack models, and as a test property for new hash functions.

2.2 The e-SPR Game

The game e-SPR is significantly harder for the attacker than c-SPR (and in particular it is not open to generic birthday attacks). It is also closer to r-SPR. There are, however, two important differences between r-SPR and e-SPR:

- The distribution from which m is chosen is uniform, but c is determined as a function of m. In particular, the joint distribution of c and m has b bits of entropy, while the pair (c, m) is $(b + n)$-bits long. This difference can in principle make e-SPR either easier or harder than r-SPR, depending on the underlying hash function.
- The attacker S gets to influence the distribution on c via its choice of the Δ_i's (but note that S must commit to the Δ_i's before it gets the random value r !)

Due to the first point above, e-SPR and r-SPR are formally incomparable, and it is possible to concoct examples of compression functions where one is easy and the other is hard (assuming r-SPR/e-SPR functions exist).

Relation of e-SPR to L-order TCR. As noted earlier, the fact that a family of compression functions is TCR does not necessarily imply that the multi-block extension of that family obtained via M-D iterations is TCR. Therefore, it is natural to search for (stronger) properties of compression functions that suffice to ensure that the M-D extension is TCR. Hong, Preneel and Lee [15] identified such a property, that they called *L-order TCR* (we recall it below), and proved that if a compression family is L-order TCR then the M-D extension is TCR on inputs of (up to) L blocks. Thus, one way to prove that our H_r construction (Eqn. (1)) is TCR is to assume that the compression function family $\{h_r\}_r$ defined by Eqn. (1) (i.e. $h_r(c, m) = h(c, m \oplus r)$) is L-order TCR. While this property may be a plausible assumption on practical compression functions (and hence can be used as evidence to support the TCR property of our H_r scheme), it turns out that our Theorem 2 (Section 3) provides for a stronger result. Indeed, that theorem only assumes h_r to be e-SPR and we show in Proposition 1 below that e-SPR is a weaker property (i.e., harder to break) of the h_r scheme than L-order TCR. (We note that this relative strength of e-SPR and L-order TCR may not hold for all hashing schemes but it does hold for our specific scheme).

Hong et al. [15] define a function family $\{h_r\}_r$ to be L-order TCR if no feasible adversary can win the following game, except with insignificant probability: As in the TCR game, the attacker commits to a message m, then it is shown the salt r and it needs to find another message m' such that $h_r(m) = h_r(m')$. The difference is that before committing to the first message m the attacker gets to learn some information about the salt r by means of L adaptive queries m_i to the hash function for which the attacker gets as a response the corresponding hash values $h_r(m_i)$.

Proposition 1 *Let $h : \{0,1\}^{b+n} \to \{0,1\}^b$ be a compression function. If the family $\{h_r\}_{r\in\{0,1\}^b}$ defined by $h_r(c, m) = h(c, m \oplus r)$ is order-L TCR(ε, t), then $h(\cdot)$ is e-SPR$(\varepsilon, L, t - O(L))$.*

Proof: (This proof is described in the "non-uniform" setting where the IV in the e-SPR game is a parameter, but it is trivial to adapt it also to the "uniform setting.)

Fix $c_0 \in \{0,1\}^n$ and let S be an e-SPR attacker against $h(\cdot)$ with respect to length L and the IV c_0, and we construct an order-L TCR attacker T against h_r. The attacker T first activates S and gets the blocks $\Delta_1, \ldots, \Delta_\ell$ for $\ell \leq L$.

For $i = 1 \ldots \ell - 1$, the attacker T queries the hash function h_r with (c_{i-1}, Δ_i), getting back $c_i \leftarrow h(c_{i-1}, \Delta_i \oplus r)$. Then T commit to $(c_{\ell-1}, \Delta_\ell)$, gets the salt r and passes it to the e-SPR attacker S. When S replies with (c', m'), T sets $c'' = c'$ and $m'' = m' \oplus r$ and outputs (c'', m'') as its colliding pair. Notice that by definition of the e-SPR game, if S is successful in its attack then for $m = \Delta_\ell \oplus r$ and $c = c_{\ell-1}$ it holds that $h(c', m') = h(c, m)$ but $(c', m') \neq (c, m)$. This means that on one hand we have

$$(c_{\ell-1}, \Delta_\ell) = (c, m \oplus r) \neq (c', m' \oplus r) = (c'', m''),$$

and on the other hand

$$h_r(c_{\ell-1}, \Delta_\ell) = h(c_{\ell-1}, \Delta_\ell \oplus r) = h(c, m) = h(c', m') = h(c'', m'' \oplus r) = h_r(c'', m'')$$

so T is also successful in its order-L TCR attack. $\square$

3 Achieving Target Collision Resistance

Recall the definition of the construction H_r from Eqn. (1),

$$H_r^c(m_1, \ldots, m_L) \stackrel{\text{def}}{=} H^c(m_1 \oplus r, \ldots, m_L \oplus r).$$

Using the games from Section 2 we can establish some relations between the corresponding SPR flavors of h and the TCR property of H_r. For this we define the TCR game for a family H_r as follows:

TCR Game: The attacker T knows the fixed IV c_0. It chooses a message M of length $L > 0$ blocks, receives a random $r \in_{\mathrm{R}} \{0,1\}^b$, and outputs a second message M' of length L'. Attacker T wins if $M' \neq M$ and $H_r^{c_0}(M') = H_r^{c_0}(M)$.

For simplicity of presentation, we first state and prove our results in the case where the TCR attacker can only provide M, M' of length an integer multiple of the block length, and it is not allowed to output M, M' such that one is a suffix of the other. This restriction on the attacker T is not significant in practice since actual implementations pads the messages and append the length to the last block, thus forcing full-block suffix-freeness. (To address the case where such length appending is not necessarily done one needs an additional "one-wayness assumption", as described in Section 3.1.1. Also, see Section 5 for a discussion on how to handle partial blocks.) Formally, we consider a modified game (denoted TCR*) where the attacker does not win the game if it violates the condition on the messages.

TCR* Game: Same as TCR but the attacker wins only if in addition to the regular TCR conditions it also holds that M, M' consists of an integral number of b-bit blocks and neither of them is suffix of the other.

Theorem 2 *(TCR Theorem, suffix-free case)*

1. *If h is* c-SPR(ε, t) *then H_r is* TCR$^*(L\varepsilon, L, t - O(L))$
2. *If h is* e-SPR(ε, L, t) *then H_r is* TCR$^*(L\varepsilon, L, t - O(L))$

This theorem is proven below. We note that these are sufficient conditions for TCR-ness *but not necessary ones.* In other words, the failure of a compression function to one of the above attacks does not necessarily mean that the induced family H_r is not TCR.[4] In particular, while the definitions of the c-SPR and e-SPR games give full freedom to the attacker in choosing the value of c', in order to break the TCR property of H_r the attacker needs to fulfill a harder task, namely, be able to "hit" the value c' with the output of the (iterated) function H (i.e., it needs to find a message P such that $c' = H^{c_0}(P)$); see Appendix B. In any case, when designing new hash functions both c-SPR and e-SPR, as well as r-SPR, should be seen as desirable design criteria.

It is also worth commenting on the tightness of the reductions in the above theorem. In both cases, c-SPR and e-SPR, there is a linear degradation of security when going from SPR to TCR. Note, however, that there is a significant difference between the *generic security* of the two games c-SPR and e-SPR. Against the former there is a trivial birthday-type generic attack while against e-SPR generic attacks achieve only linear advantage. Thus, the reduction from e-SPR shows a worst-case "quadratic degradation" generic TCR attack against H_r (and not "cubic degradation" as the reduction cost in the c-SPR to TCR case could indicate). Moreover, TCR attacks with quadratic degradation (i.e., with success probability in the order of $tL/2^n$) against H_r are indeed possible since SPR attacks against H can be translated into TCR attacks against H_r and we know that such birthday-type SPR attacks against H exist [12, 17].

This motivates two questions: Can we have a flavor of SPR for which there is a tight reduction to TCR? Can this SPR game be defined such that it is only vulnerable to linear generic attacks? The answer to the first question is YES: in Appendix A.1 we present such a game, called m-SPR (m for multiple), and its tight reduction to TCR. The answer to the second question is obviously NO, since as said SPR birthday attacks against H_r do exist. It is possible, however, to design variants of M-D hash functions (e.g., appending a sequence number to each block or using the "dithering" technique proposed by Rivest [24]) for which the best generic attacks achieve linear degradation only. Thus, the main motivation and usefulness of the game m-SPR and its tight reduction to TCR is for the analysis of such variants.

3.1 Proof of Theorem 2

In the proof below we assume that the attacker T never outputs M, M' where one is the suffix of the other. We also make the convention that when algorithm A calls another algorithm B and B aborts then A aborts too.

Lemma 3 *If h is* c-SPR(ε, t) *then H_r is* TCR$^*(L\varepsilon, L, t - O(L))$.

Before proving this lemma, we comment that the exact assertion of this lemma (and thus the proof) depend on whether we view the IV c_0 as a random input to the TCR attacker or a parameter of the

[4] Recent work by Lisa Yin [35] has shown a concrete interesting example: while current attacks show that the compression function of MD5 is neither e-SPR nor r-SPR, these attacks do not seem to apply to breaking the TCR property of H_r when instantiated with MD5.

TCR game (cf. comment on page 7). In the reduction below we suppress c_0 with the understanding that if it is a parameter then the same parameter is used in the reduction, and if it is an input then S chooses c_0 at random and gives it to T.

Proof: Given TCR attacker T we build c-SPR attacker S:

1. S gets input m, invokes T and gets from T the first message $M = (m_1, \ldots, m_L)$.
2. S chooses $\ell \in_{\mathrm{R}} \{1, \ldots, L\}$, sets $r = m \oplus m_\ell$, and returns r to T.
3. T outputs the second message $M' = (m'_1, \ldots, m'_{L'})$ or $\perp$.
4. S checks that there exists $\ell' \in \{1, \ldots, L'\}$ such that (1) either $m'_{\ell'} \neq m_\ell$ or $H_r(m'_1, \ldots, m'_{\ell'-1}) \neq H_r(m_1, \ldots, m_{\ell-1})$; and (2) $H_r(m'_1, \ldots, m'_{\ell'}) = H_r(m_1, \ldots, m_\ell)$. If no such ℓ' exists then S aborts.
5. Otherwise S outputs $c = H_r(m_1, \ldots, m_{\ell-1})$, $c' = H_r(m'_1, \ldots, m'_{\ell'-1})$ and $m' = m'_{\ell'} \oplus r$. (If there is more than one index with the properties from above then S chooses ℓ' arbitrarily among all these indexes.)

Turning to the analysis of S, we first observe that if T does not abort then there must exist a pair of indexes i, i' that satisfy the properties from Step 4, namely $H_r(m'_1, \ldots, m'_{i'}) = H_r(m_1, \ldots, m_i)$ and either $m'_{i'} \neq m_i$ or $H_r(m'_1, \ldots, m'_{i'-1}) \neq H_r(m_1, \ldots, m_{i-1})$.

To see that, note that since M, M' are suffix-free there exists an integer $j < \min(L, L')$ such that $m_{L-j} \neq m'_{L'-j}$. Consider the smallest such j (i.e., the last block where M, M' differ when their ends are aligned). Also, let k be smallest non-negative integer such that $H_r(m'_1, \ldots, m'_{L'-j+k}) = H_r(m_1, \ldots, m_{L-j+k})$ (i.e., the first "collision" after these "last differing blocks" in the computation of $H_r(M'), H_r(M)$). Clearly, such k exists since T did not abort and so there must be some collision because $H_r(M') = H_r(M)$.

Now setting $i = L - j + k$ and $i' = L' - j + k$ we get the pair that we need. This is because we have $H_r(m'_1, \ldots, m'_{L'-j+k}) = H_r(m_1, \ldots, m_{L-j+k})$ by definition, and (a) either $k = 0$ in which case $m_{L-j+k} \neq m'_{L'-j+k}$, or (b) $k > 0$ in which case $H_r(m'_1, \ldots, m'_{L'-j+k-1}) \neq H_r(m_1, \ldots, m_{L-j+k-1})$.

Next, we argue that the random choice of ℓ is independent of the transcript that T sees, and in particular it is independent of the values of any such pair i, i' as above. This is easy to see, since the distributions of all the variables in the execution of S remains unchanged if instead of choosing m at random and setting $r = m_\ell \oplus m$ we choose r at random and set $m = m_\ell \oplus r$. In the new game, however, we can postpone the choice of ℓ (and m) until the end, and so it is clearly independent of M, M'.

It follows that whenever T does not abort, there exists a pair i, i' as above, and in this case the probability that S chooses $\ell = i$ is (at least) $1/L$. If that happens then there exists ℓ' as needed (i.e., $\ell' = i'$) so S does not abort. It remains to show that in this case S wins the c-SPR game. This follows since by the condition on ℓ, ℓ' we have:

$$\text{either } c = H_r(m_1, \ldots, m_{\ell-1}) \neq H_r(m'_1, \ldots, m'_{\ell'-1}) = c' \quad \text{or} \quad m' = m'_{\ell'} \oplus (m_\ell \oplus m) \neq m$$

and also (due to $m = r \oplus m_\ell$ and $m' = r \oplus m'_{\ell'}$):

$$h(c, m) = h(c, r \oplus m_\ell) = H_r(m_1, \ldots, m_\ell) = H_r(m'_1, \ldots, m'_{\ell'}) = h(c', r \oplus m'_{\ell'}) = h(c', m')$$

□

The attacker in the proof of Lemma 3 makes little use of the full freedom provided by the c-SPR game in (adaptively) choosing c. Indeed the value of c used by S is set as soon as S chooses ℓ, and this choice is *independent of* m. This motivates the e-SPR game where, by definition, S can influence c only through the choice of values Δ to which it commits before seeing r. Thus, e-SPR represents a better approximation of the TCR game, by giving the attacker in e-SPR less artificial freedom than in c-SPR, and making it much closer to SPR (in particular, by disallowing generic birthday attacks against it). This makes the following lemma much stronger than Lemma 3 though its proof follows the same lines. Note that we are still paying a linear (in L) degradation of security but this time the reduction is to a "non-birthday" problem. (See also the discussion about security degradation preceding Section 3.1.)

Lemma 4 *If h is* e-SPR(ε, L, t) *then H_r is* TCR$^*(L\varepsilon, L, t - O(\ell))$.

Proof: This is a simple adaptation of the proof of Lemma 3. Given TCR attacker T we build e-SPR attacker S:

1. S invokes T and gets from T the first message $M = (m_1, \ldots, m_L)$.
2. S chooses $\ell \in_{\mathrm{R}} \{1, \ldots, L\}$ and sets $\Delta_i = m_i$ for $i = 1, \ldots, \ell$.
3. S receives a random r and forwards it to T. (The values of c, m are evaluated as $c = H_r(\Delta_1, \ldots, \Delta_{\ell-1}) = H_r(m_1, \ldots, m_{\ell-1})$ and $m = \Delta_\ell \oplus r = m_\ell \oplus r$.)
4. T outputs the second message $M' = (m'_1, \ldots, m'_{L'})$ or $\perp$.
5. S checks that there exists $\ell' \in \{1, \ldots, L'\}$ such that (1) either $m'_{\ell'} \neq m_\ell$ or $H_r(m'_1, \ldots, m'_{\ell'-1}) \neq H_r(m_1, \ldots, m_{\ell-1})$; and (2) $H_r(m'_1, \ldots, m'_{\ell'}) = H_r(m_1, \ldots, m_\ell)$. If no such ℓ' exists then S aborts.
6. Otherwise S outputs $c' = H_r(m'_1, \ldots, m'_{\ell'-1})$ and $m' = m'_{\ell'} \oplus r$. (If there is more than one index with the properties from above then S chooses ℓ' arbitrarily among all these indexes.)

The analysis of S is nearly identical to the analysis in the proof of Lemma 3. The only difference is that here it is even easier to argue that the choice of ℓ is independent of the transcript of T. □

(We again note that the lemma above and its proof can be interpreted in two ways, depending on whether c_0 is an input or a parameter. The exact same proof applies to both cases, however, so we suppressed c_0 above.)

3.1.1 Extension to non-suffix-free messages

We stated Theorem 2 and its proof in terms of the TCR* game, namely, the TCR game in which the attacker is restricted to suffix-free pairs M, M'. For the sake of generality, we extend these results here to the case where inputs to H_r may not be suffix free. For this extension we need the following one-wayness assumption.

Assumption 5 (OWH**)** *The assumption* OWH *for the compression function h asserts that given a random c, it is hard to find a non-null message M such that $H^c(M) = c$.*

We use our usual notation $\mathrm{OWH}(\varepsilon, t)$ to denote the assumption that no t-time attacker can violate the assumption OWH with probability better than ε.

Some comments are in order here. First, note that this assumption is stated in the "uniform" setting where the IV c is chosen at random. Its "non-uniform" counterpart (with respect to a parameter c_0) does not make formal sense, for the same reason that the assertion "SHA256 as per FIPS-180-2 is collision resistant" does not make formal sense.[5] This means that formally, Theorem 6 below only applies to this "uniform" setting.[6]

Second, the above assumption is clearly implied by the assumption that for a random c it is hard to find c', m such that $h(c', m) = c$. Furthermore, under a very mild condition on the structure of h, this last assumption is equivalent to the standard assumption that h is a one-way function (i.e., for random c, m it is hard given $h(c, m)$ to find c', m' such that $h(c', m') = h(c, m)$). Specifically, the condition that we need is that for random c, m, the distribution over $h(c, m)$ is statistically close to (or at least indistinguishable from) the uniform distribution on $\{0, 1\}^n$.

It is well known that one-wayness is implied by second-preimage resistance, and is it easy to verify that the same holds also for our notions of c-SPR and e-SPR (the latter in its "uniform" interpretation where the IV is random). It follows that under that mild structural condition, the assumption OWH is implied by the assumptions e-SPR and c-SPR and so is redundant in Theorem 6 below. Still, we prefer to state the theorem with this assumption since (a) it makes it easier to understand, and (b) it applies also to functions h for which that structural condition does not hold.

Theorem 6 *(TCR Theorem , non-suffix-free case)*
1. If h is $\text{c-SPR}(\varepsilon, t)$ *and* $\mathrm{OWH}(\varepsilon', t)$ *then H_r is* $\mathrm{TCR}(\varepsilon' + L\varepsilon, L, t - O(L))$.
2. If h is $\text{e-SPR}(\varepsilon, L, t)$ *and* $\mathrm{OWH}(\varepsilon', t)$ *then H_r is* $\mathrm{TCR}(\varepsilon' + L\varepsilon, L, t - O(L))$.

The proof is a small adaptation of the proofs of lemmas 3 and 4. One just observes that in the case where one of M, M' is a proper suffix of the other, then either the prefix of the longer message violates the one-wayness assumption (i.e., we have $H^c(\text{prefix}) = c$), or else the same analysis from lemmas 3 and 4 applies. The reason that ε' is not multiplied by L in the expressions above is that in the former case we do not care about the value of ℓ that was chosen at the beginning of the reduction. (See the proof of Theorem 7 for the formal argument.)

4 Enhanced Target Collision Resistance

Recall the definition of the construction $\tilde{H}_r$ from Eqn. (2),

$$\tilde{H}^c_r(M) \overset{\text{def}}{=} H^c_r(0|M) = H^c(r, m_1 \oplus r, \ldots, m_L \oplus r)$$

We show that under the same c-SPR and e-SPR assumptions, the construction $\tilde{H}_r$ is enhanced TCR (eTCR). We start by defining the eTCR game.

[5]We could use the same "solution" here as is sometimes used to argue about collision-resistance, assuming that H^{c_0} itself is already a random member of some imaginary family of functions. We chose not to use this "solution" because we think it is ugly and we don't really need it.

[6]However, the reduction that proves Theorem 6 is meaningful also for the case of fixed IV, showing a constructive transformation from TCR attack to either an SPR attack or to an attack that violates the one-wayness assumption.

eTCR Game: Attacker T chooses a message M of length $L \geq 0$ blocks, receives a random $r \in_{\mathrm{R}} \{0,1\}^b$, and outputs another $r' \in \{0,1\}^b$ and a message M' of length L' blocks. Attacker T wins if $(M', r') \neq (M, r)$ and $H^{c_0}_{r'}(M') = H^{c_0}_r(M)$.

Note that the attacker can win this game even when $M = M'$, so long as $(M', r') \neq (M, r)$. We remark that as opposed to the case of the construction H_r and the standard TCR game, here assuming suffix-freeness does not help us, because the attacker can specify $r' \neq r$ and so even if M, M' are suffix free the messages $M \oplus r$, $M' \oplus r'$ perhaps are not. (However, we can go back to the suffix-free case by using length padding outside of the randomness, as discussed in Section 5.) Below we therefore use the OWH assumption as in Theorem 6, thus getting:

Theorem 7 (eTCR security)

1. *If h is* c-SPR(ε, t) *and* OWH(ε', t) *then $\tilde{H}_r$ is* eTCR$(\varepsilon' + (L+1)\varepsilon, L, t - O(L))$.

2. *If h is* e-SPR$(\varepsilon, L+1, t)$ *and* OWH(ε', t) *then $\tilde{H}_r$ is* eTCR$(\varepsilon' + (L+1)\varepsilon, L, t - O(L))$.

Proof: Below we only prove part 2. Part 1 can be obtained as a corollary, since an e-SPR attacker can be trivially converted to a c-SPR attacker with the same success probability. The proof is a small adaptation of the proof of Lemma 4. The idea is that S sets $\Delta_0 = 0$ (and the other Δ_i's as before) and then uses r' instead of r to compute c' and m'. In more details, given an enhanced-TCR attacker T be build e-SPR attacker S:

1. On input an IV c, S invokes $T(c)$ and gets from T the first message $M = (m_1, \ldots, m_L)$.

2. S sets $m_0 = 0$, chooses $\ell \in_{\mathrm{R}} \{0, \ldots, L\}$ and sets $\Delta_i = m_i$ for $i = 0, \ldots, \ell$.

3. S receives a random r and forwards it to T. (Also c, m are evaluated as $c = H^c_r(\Delta_0, \Delta_1, \ldots, \Delta_{\ell-1}) = H^c_r(0, m_1, \ldots, m_{\ell-1})$ and $m = \Delta_\ell \oplus r = m_\ell \oplus r$.)

4. T outputs either the second salt r' and message $M' = (m'_1, \ldots, m'_{L'})$ or $\perp$.

5. S sets $m'_0 = 0$, and searches for an index $\ell' \in \{0, \ldots, L'\}$ such that (1) either $m'_{\ell'} \oplus r' \neq m_\ell \oplus r$ or $H^c_{r'}(0, \ldots, m'_{\ell'-1}) \neq H^c_r(0, m_1, \ldots, m_{\ell-1})$; and (2) $H^c_{r'}(0, m'_1, \ldots, m'_{\ell'}) = H^c_r(0, m_1, \ldots, m_\ell)$. If no such ℓ' exists then S aborts.

6. Otherwise S outputs $c' = H^c_{r'}(0, m'_1, \ldots, m'_{\ell'-1})$ and $m' = m'_{\ell'} \oplus r'$. (If there is more than one index with the properties from above then S chooses ℓ' arbitrarily among all these indexes.)

To analyze S, let SUFF denote the event in which one of the messages

$$X = (r, m_1 \oplus r, \ldots, m_L \oplus r) \text{ and } X' = (r', m'_1 \oplus r', \ldots, m'_{L'} \oplus r')$$

in the game above is a suffix of the other, and in addition hashing only the prefix of the longer message yields back the IV c. In other words, the event SUFF occurs whenever we have either $X' = Y|X$ or $X = Y|X'$ with some $Y \neq \Lambda$ for which $H^c(Y) = c$.

Let $\varepsilon_{\text{suff}}$ be the probability of the event SUFF, and let ε^* be the probability of the event in which T succeeds but the event SUFF does not happen. In the latter case we can apply the same analysis as in Lemma 4, showing that there must exist a pair of indexes i, i' that satisfy the conditions in step 5, that with probability at least $1/(L+1)$ we have $\ell = i$, and if that happens then S succeeds.

The only part of the analysis that is slightly different than in the proof of Lemma 4 is proving the existence of the indexes i, i' in the case where one of X, X' is a prefix of the other but $H^c(Y) \neq c$. In this case, denote by $X|_i$ the $(i+1)$-block prefix of X (i.e., blocks 0 through i), and similarly denote $X'|_i$. Let j be the smallest integer such that $H^c(X|_{L-j}) \neq H^c(X'|_{L'-j})$ (i.e., the last place where the chaining values do not agree). Since $H^c(Y) \neq c$ then there exists such index $j \leq \min(L+1, L'+1)$, and on the other hand $j > 0$ since the success of T implies that $H^c(X_L) = H^c(X'_{L'})$. Then setting $i = L - j + 1$ and $i' = L' - j + 1$ we get the pair of indexes that we need.

We thus conclude that the success probability of S is at least $\varepsilon^*/(L+1)$. Since (by definition) the success probability of T is at most $\varepsilon^* + \varepsilon_{\text{suff}}$, we can complete the proof by showing a OWH-attacker with success probability $\varepsilon_{\text{suff}}$. Such attacker S' is obvious: It gets c as input and runs the same execution as the e-SPR attacker S from above (choosing itself the random r), and at the end if the event SUFF occurs then it outputs the prefix Y. $\square$

5 Using Randomized Hashing in Signatures

The randomized hashing modes presented in this paper are intended to replace the deterministic hashing used by standardized signature schemes such as RSA [22] and DSS [11]. As shown throughout the paper, this replacement frees these schemes from their current essential dependency on full collision resistance by basing their security on significantly harder to break properties of the underlying hash functions. Of the two schemes analyzed here, the $\tilde{H}_r$ mode (Eqn. (2)) is best suited for this task as it does not require the explicit signing of the salt r and hence allows for more implementation flexibility. Therefore, we propose $\tilde{H}_r$ as the preferred mode for practical use and provide a full, detailed, specification, named RMX, ready for standardization in Appendix D. (See also [7] for some implementation experience and a demonstration of the viability of such a solution in practice.) In the remainder of this section we discuss two general approaches for the integration of $\tilde{H}_r$ with actual signature schemes.

The two main approaches depend on whether an application can accommodate the sending of the salt r in addition to the message and signature or whether any increase on the size of the information is not possible. The first case is simpler and requires the least changes to standards. Most applications will be able to send the the extra salt, especially that r needs not be too long, say in the range of 128-256 bits (see below). Examples of such applications are digital certificates, code signing, XML signatures, etc. In this case, upon the need to sign a message M the modified application will: (i) choose a random salt r; (ii) transform the message $M = (m_1, \ldots, m_L)$ into the message $M' = (r, m_1 \oplus r, \ldots, m_L \oplus r)$; (iii) invoke the *existing* signature operation (including the existing hash operation) on M'; (iv) when the message M and its signature are to be transmitted, transmit r, M and the computed signature (on M'). The verification side computes M' from the received r and M and applies its *existing* signature verification operation on M'. This approach allows for preserving the existing processing order (including a one-pass over the signed message[7]) and the possible pre-computation (ahead of signature generation) of the pair $(r, \tilde{H}_r(M))$.

Note that in this case the application can use its existing signature and verification processing (whether in software, hardware or both) *without any change.* In this case, signature standards, such as [22, 11], that follow the hash-then-sign paradigm, need no change except for adding the "front end" message randomization (to generate M').

[7]Note that steps (ii)-(v) can be done in parallel, namely, on a block-per-block processing compatible with the way many of the existing implementations of hash functions work (this is the case in both signing and verifying ends).

Applications where an increase in the size of messages or signatures is impractical will need to resort to a different approach: either re-use the existing randomness (in the case of probabilistic signatures) or encode the salt r "under the signature". The former is the case for the probabilistic schemes DSS [11] and RSA-PSS [22] where the already existing random values r (used internally by the PSS operation in the case of RSA-PSS and as the signature component $r = g^k$ in the case of DSS) can be re-used as the hashing salt and thus require no additional transmission. The latter case, i.e., encoding r "under the signature", is the case of the traditional deterministic RSA encoding of PKCS#1 v1.5 [22]. In this case, instead of padding the (deterministic) hash value to a full modulus size as done today, one will pad the concatenation of r and $\tilde{H}_r(M)$ to a full modulus size and then apply to it the private RSA operation. In this case, the salt r can be retrieved by the verifying party using the RSA public operation and hence no extra transmission is required.

Evaluating the complexity of performing the above changes depends on particular settings and applications. Clearly, no change to existing applications, signature modules or standards is "too simple" and one cannot ignore the cost of engineering the above changes. However, considering that applications that compute digital signatures will be upgraded in any case to use new hash functions, one should use this opportunity to also upgrade to randomized hashing. In particular, it is worth noting that many difficult issues for applications such as preserving backwards compatibility, the signaling/negotiation of algorithms and capabilities, version rolling, etc. are not made worse by our proposal than what is already needed to support a simple hash function upgrade [4]. Similarly, our implementation experience [7] shows that the extra complexity added by our mode relative to the adoption of a new hash function is not very significant (except for the ability to generate randomness at the signer's end, something most modern cryptographic schemes provide). Considering the simplicity and minimalistic nature of our randomization scheme, we believe that the extra changes, randomness generation and transmission of r, are well worth the substantial security gain they provide to existing and future signature schemes.

Specification Details. As said, a concrete and detailed specification of the $\tilde{H}_r$ mode is presented in Appendix D. Here we mention two elements that we omitted so far and that need to be included in a practical instantiation.

Shorter salt. The construction $\tilde{H}_r$ as defined in Eqn. (2) uses salt of length b, i.e. one message block. In practical instantiations, we propose to choose a salt string r' of length between 128 and b, and create the b-bit salt r by concatenating r' with itself as needed, possibly with the last r' repetition being truncated (e.g., in the case of $b = 512$ and a 160-bit r', one would have $r = r'||r'||r'||r''$ where r'' represents the first 32 bits of r'). The analysis in this paper applies to this modified salt, except that now the assumptions c-SPR, e-SPR are stated in terms of random messages distributed according to the distribution induced by r'. Of course, the plausibility of these assumptions needs to be evaluated; however, as long as r' is not too short, these modified SPR-type assumptions seem very reasonable (it is even possible that the added "structure" in these repetitions makes the cryptanalytical problem more difficult).

Last block padding. The Merkle-Damgård construction pads the input message to an integral number of blocks and includes an encoding of the length of the original message in the last block (thus ensuring a suffix-free message space). The analysis in this work assumes that the salt r is xor-ed to the padded message, but in practice it is likely that the xor will happen first (as part of the randomized mode-of-operation) and the padding will then be applied to the randomized message (as part of the underlying hash function). Simply switching the order and using randomize-then-pad may mean that the last block is only randomized in its first few bits. A more robust alternative is to use two levels of padding. Namely, we first use suffix-free padding of the last block to a full block minus 64 bits (i.e., 448 bits), then xor the salt to this padded message, and finally apply the original hashing scheme (which will include the length of the padded message in the remaining 64 bits). See Appendix D.

6 Open Problems

In light of the results in this paper, we feel that more focus should be placed on evaluating current and future hash functions against the c-SPR and e-SPR attack scenarios. Specifically, we would like to offer the c-SPR scenario as a "toy example" for developing cryptanalytical tools that may prove useful in assessing randomized hashing, and then study the e-SPR scenario as a stronger foundation for our scheme. Other open problems that arise from this work are briefly discussed next.

A different construction for eTCR. Another natural proposal for obtaining enhanced-TCR is to define $\overline{H}^c_r(M) = H^c(r|H^c_r(M))$. It may be interesting to study this variant and see what advantages (and disadvantages) it offers vis-a-vis the construction $\tilde{H}$ from Eqn. (2).

The random-oracle model and weak hash functions. One inherent difficulty in formally proving the security of TCR and eTCR schemes in the context of DSS and RSA as specified in the official standards [22, 11] is that these schemes do not have a proof of security (not even in the case that the underlying hash function is fully collision resistant). The "closest relatives", namely, the DSA-II variant of Brickel, Pointcheval, Vaudenay and Yung [8] or the RSA-PSS scheme of Bellare and Rogaway [2], have a proof of security in the random oracle model. Interestingly, these proofs use in an essential way the randomization of the hash function, not unlike the TCR or eTCR constructions. In some sense one can think, for example, of the use of DSS with a eTCR scheme as an "instantiation" of DSS-II. The obvious question is: how can we state anything related to random oracles when we are dealing with relatively weak (at least not CR) hash functions (clearly, random oracles are "very strong hash functions"). See some results related to this question in Appendix C and in [20].

We also point out that the random-oracle proofs for some of the above (randomized) signature schemes do not essentially use the collision resistance property of the random-oracle (as evidenced, for example, by the fact that some of these proofs, such as those following [23], remain meaningful even when you have a random-oracle with relatively short output, e.g., 80 bits). This brings up the interesting question of exhibiting variants of the random-oracle model where one can argue about functions that "behave randomly but are not collision-resistant" (e.g., a random oracle H with an associated oracle that outputs random H collisions upon request).

Acknowledgments. We thank Ran Canetti, Orr Dunkelman, Charanjit Jutla, Burt Kaliski, Tom Shrimpton and Lisa Yin for helpful discussions. We are particularly indebted to Lisa for her many insightful comments.

References

[1] Mihir Bellare, Ran Canetti, Hugo Krawczyk, "Keying Hash Functions for Message Authentication". CRYPTO 1996. 1-15

[2] Mihir Bellare and Phillip Rogaway, "The Exact Security of Digital Signatures – How to Sign with RSA and Rabin", EUROCRYPT 96.

[3] Mihir Bellare and Phillip Rogaway, "Collision-Resistant Hashing: Towards Making UOWHFs Practical", CRYPTO 97, LNCS 1294, 1997

[4] Steven M. Bellovin and Eric K. Rescorla, "Deploying a New Hash Algorithm", NDSS'06. `http://www.cs.columbia.edu/~smb/papers/new-hash.pdf`

[5] Eli Biham and Rafi Chen, "Near-Collisions of SHA-0", CRYPTO 2004.

[6] Eli Biham, Rafi Chen, Antoine Joux, Patrick Carribault, Christophe Lemuet and William Jalby, "Collisions of SHA-0 and Reduced SHA-1", EUROCRYPT 2005.

[7] D. Boneh, S. Halevi, H. Krawczyk, M. McIntosh, and W. Shao, "Implementation of the Halevi-Krawczyk Randomized Hashing Scheme", submitted.

[8] Ernest Brickell, David Pointcheval, Serge Vaudenay, and Moti Yung, "Design and Validations for Discrete Logarithm Based Signature Schemes", PKC'2000.

[9] Florent Chabaud and Antoine Joux, "Differential Collisions in SHA-0", CRYPTO 98.

[10] Ivan Damgård, "A design principle for hash functions", CRYPTO 1989.

[11] Digital Signature Standard (DSS), FIPS 186, May 1994.

[12] Richard Drews Dean, "Formal Aspects of Mobile Code Security", Ph.D Dissertation, Princeton University, January 1999.

[13] Shafi Goldwasser, Silvio Micali and Ronald L. Rivest, "A Digital Signature Scheme Secure Against Adaptive Chosen-Message Attacks", *SIAM J. Comput.* 17(2): 281-308 (1988)

[14] Shai Halevi and Hugo Krawczyk, "The RMX Transform and Digital Signatures", 2nd NIST Hash Workshop, August 2006. See `http://www.ee.technion.ac.il/~hugo/rhash/`

[15] Deukjo Hong, Bart Preneel and Sangjin Lee, "Higher Order Universal One-Way Hash Functions", ASIACRYPT 2004.

[16] Antoine Joux, "Multicollisions in Iterated Hash Functions. Application to Cascaded Constructions", CRYPTO 2004.

[17] John Kelsey and Bruce Schneier, "Second Preimages on n-Bit Hash Functions for Much Less than 2^n Work", EUROCRYPT 2005.

[18] Ralph Merkle, "One way hash functions and DES", CRYPTO 1989.

[19] Ilya Mironov, "Hash Functions: From Merkle-Damgård to Shoup", EUROCRYPT 2001, 166-181.

[20] Ilya Mironov, “Collision-Resistant No More: Hash-and-Sign Paradigm Revisited”, Public Key Cryptography 2006, 140-156.

[21] Moni Naor and Moti Yung, “Universal One-Way Hash Functions and their Cryptographic Applications”, STOC 1989.

[22] PKCS #1 v2.1: RSA Cryptography Standard, RSA Laboratories, June 14, 2002

[23] David Pointcheval and Jacques Stern, “Security Arguments for Digital Signatures and Blind Signatures”, *J.Cryptology* (2000) 13:361-396.

[24] Ron Rivest, “Abelian square-free dithering for iterated hash functions”, Presented at ECrypt Hash Function Workshop, June 21, 2005, Cracow.

[25] Phillip Rogaway, Thomas Shrimpton, “Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance”. FSE 2004, 371-388.

[26] John Rompel, “One-way functions are necessary and sufficient for secure signatures”, STOC 1990, pp. 387-394.

[27] Victor Shoup, “A Composition Theorem for Universal One-Way Hash Functions”, EUROCRYPT 2000.

[28] Dan Simon, “Finding collisions on a one-way street: Can secure hash functions be based on general assumptions?”, EUROCRYPT 98, pp. 334-345.

[29] Michael Szydlo and Yiqun Lisa Yin, “Collision-Resistant usage of MD5 and SHA-1 via Message Preprocessing”, CT-RSA 2006, pp. 99-114.

[30] W3 Consortium. “XML Advanced Electronic Signatures (XAdES)”, W3C Note 20 February 2003. `http://www.w3.org/TR/XAdES/`

[31] Xiaoyun Wang, Xuejia Lai, Dengguo Feng, Hui Chen, and Xiuyuan Yu, “Cryptanalysis of the Hash Functions MD4 and RIPEMD”, EUROCRYPT 2005.

[32] Xiaoyun Wang and Hongbo Yu, “How to Break MD5 and Other Hash Functions”, EUROCRYPT 2005.

[33] Xiaoyun Wang, Hongbo Yu, and Yiqun Lisa Yin, “Efficient Collision Search Attacks on SHA-0”, CRYPTO 2005.

[34] Xiaoyun Wang, Yiqun Lisa Yin, and Hongbo Yu “Finding Collisions in the Full SHA-1”, CRYPTO 2005.

[35] Yiqun Lisa Yin, NIST 2nd Hash Workshop, August 2006. `http://www.csrc.nist.gov/pki/HashWorkshop/2006/Presentations/YIN_NIST2ndHashWorshop-ContiniYin-Aug25-2006.pdf`

A More Proofs

A.1 From m-SPR to TCR*

In this section we introduce a new SPR game, called m-SPR (m for multiple), whose main advantage over previous games is that it has a tight (constant factor) reduction to TCR. Thus, no TCR attack can do better against H_r than what an m-SPR attack can do. In the case of H_r as we defined, the m-SPR game is still vulnerable to birthday-type attacks (i.e., with winning probability $tL/2^n$). However, if one modifies H_r (or the underlying hash function H) to avoid such attacks against the m-SPR game then one immediately obtains the same lower bound on the TCR security of such family. Specific examples of such potential modification were discussed earlier e.g., the variant that adds a sequence number to each block of input. Under this variant the best generic attack against the m-SPR game achieves a winning probability of at most $O(t)/2^n$ and therefore the same is true for any generic TCR attack against such modified H_r. The m-SPR game is defined as follows:

1. Attacker S knows the IV c_0. It chooses an integer $\ell > 0$ and ℓ b-bit blocks $\Delta_1, \ldots, \Delta_\ell$.
2. A random $r \in_{\mathrm{R}} \{0,1\}^b$ is chosen and for $i = 1, \ldots, \ell$ we set $m_i = r \oplus \Delta_i$ and $c_i = H(r \oplus \Delta_1, \ldots, r \oplus \Delta_{i-1})$.
3. S outputs c' and m'.
4. S wins if for some $i \in \{1, \ldots, \ell\}$, $h(c_i, m_i) = h(c', m')$ and $(c_i, m_i) \neq (c', m')$.

Note that in spite of the tight reduction in the next lemma, TCR and m-SPR are not equivalent since in the TCR game it is not enough to find *any* c' but one faces the harder problem of finding some M_0 for which $c' = H_r(M_0)$ (hence, even a break of m-SPR does not necessarily imply a break of TCR).

Lemma 8 *If h is* m-SPR(ε, L, t) *then H_r is* TCR$^*(\varepsilon, L, t - O(L))$

Proof: The reduction from TCR to m-SPR is similar to the proof of Lemma 4 with some straightforward (but significant) changes. For completeness we show this reduction here. Given TCR attacker T we build m-SPR attacker S:

1. S invokes T and receives the first message $M = (m_1, \ldots, m_L)$.
2. S sets $\Delta_i = m_i$ for $i = 1, \ldots, L$.
3. S gets a random r and forwards it to T.
4. T outputs $M' = (m'_1, \ldots, m'_{L'})$ or $\perp$.
5. For $i = 1, \ldots, L$, S sets $c_i = H(r \oplus \Delta_1, \ldots, r \oplus \Delta_{i-1})$. Then S checks that there is a pair i, i' $(1 \leq i \leq L'$ and $1 \leq i' \leq L')$ such that
   (1) $H_r(m'_1, \ldots, m'_{i'}) = H_r(m_1, \ldots, m_i)$, and
   (2) either $m'_{i'} \neq m_i$ or $H_r(m'_1, \ldots, m'_{i'-1}) \neq H_r(m_1, \ldots, m_{i-1})$.
   If no such i, i' exist then S aborts.

6. Otherwise S outputs $c' = H_r(m'_1, \ldots, m'_{i'-1})$ and $m' = m'_{i'} \oplus r$ which collide with the pair (c_i, m_i).

$\square$

B A Stronger Result: The hit-SPR Assumption

As pointed out in Section 3 the c-SPR and e-SPR definitions were chosen as being relatively simple and natural assumptions on the compression function that suffice to guarantee the security of our TCR and eTCR schemes. On the other hand, these assumptions do not fully capture the hardness of breaking our schemes. Indeed, it is possible that one may break a compression function under either the e-SPR or c-SPR games and the H_r and $\tilde{H}_r$ schemes still be TCR and eTCR, respectively. One timely example is MD5. Lisa Yin [35] has observed that the compression function of MD5 is not r-SPR or e-SPR. This however does not directly translate into attacks against our schemes if instantiated using MD5. The point is that to break these schemes the attacker needs to solve a problem that is harder than e-SPR. That is, while the definition of the e-SPR game gives full freedom to the attacker in choosing the value of c', to break H_r as TCR or $\tilde{H}_r$ as eTCR requires the attacker to be able to "hit" the value c' with the output of the (iterated) function H on some known message P. This is captured in the following modification of the e-SPR game:

hit-SPR: The game is parametrized by an IV c_0. S chooses $\ell \geq 1$ values $\Delta_i,\ i = 1, \ldots, \ell$, each of length b bits; then S receives a random $r \in \{0,1\}^b$ and c, m are set to $m = r \oplus \Delta_\ell$ and $c = H^{c_0}(r \oplus \Delta_1, \ldots, r \oplus \Delta_{\ell-1})$. S chooses P, m', where m' is of length b and P is a message of arbitrary length. S wins the game if $h(c, m) = h(c', m')$ where $c' = H^{c_0}(P)$.

As far as current knowledge goes, the compression function of MD5 may still be hit-SPR even though it is not r-SPR or e-SPR, thus showing the non-triviality of breaking our schemes even in the presence of serious weaknesses in the hash function (this is not to say that one should use MD5 when building practical instantiations of our schemes as stronger hash functions are available).

The sufficiency of the hit-SPR assumption for the security of our schemes is captured by the following theorem whose proof follows from the proofs of Theorems 6 and 7.

Theorem 9

1. *If h is* hit-SPR(ε, L, t) *and* OWH(ε', t) *then H_r is* TCR$(\varepsilon' + L\varepsilon, L, t - O(L))$.
2. *If h is* hit-SPR$(\varepsilon, L+1, t)$ *and* OWH(ε', t) *then $\tilde{H}_r$ is* eTCR$(\varepsilon' + (L+1)\varepsilon, L, t - O(L))$.

C Relying on strong unforgeability of DSA-like signatures

Recall that the traditional approach for using TCR hashing for signature is to rely on the underlying signature scheme to also "sign the salt". In this work we have circumvented this need by developing a stronger, enhanced-TCR, randomization scheme where such signature is not required. For the sake of completeness, we try to clarify here the security properties that we need from DSA-like signature schemes if we want to use them with the TCR construction H_r from Eqn. (1) rather than the enhanced-TCR construction $\tilde{H}_r$ from Eqn. (2). Our analysis goal in this section is very modest: to prove that IF the DSA signature scheme is secure when using a deterministic hash function H then switching to the randomized H_r does not break anything. Moreover, we do not know if DSA achieves the properties required by our analysis below (e.g., strong unforgeability) and we do know that other ElGamal-type signatures do not satisfy them. Hence we recommend to use the enhanced-TCR solution in practice. The interested reader should also consult the work by Mironov [20] that addresses a question similar to the one discussed in this appendix. Two differences with the analysis presented in this section is that Mironov analyzes variants of the DSS and RSA-PSS (rather than the standardized schemes) and that his analysis is carried in the random oracle model. While we try to avoid the use of the random oracle model, we note that the strong unforgeability requirements below are easy to achieve in this model.

The class of "DSA-like" signatures that we consider here consists of schemes where the signature is a pair (r, s) with r random and s depending only on the hash of the message. In more details, to compute a signature on M we first hash it down to get $\eta = H(M)$, then we choose at random some string ρ and compute $r = f(\rho)$ and $s = g(\rho, \eta, \mathsf{sk})$, where f, g are some functions and sk is the secret signing key.

The intuition behind the results in this section is that if a signature scheme like this is *strongly existentially unforgeable* then one should be able to simply use the same r also as the randomness for the hashing. Recall that a scheme is strongly existentially unforgeable if the attacker not only cannot forge a signature on a new message, but cannot even come up with a different valid signature on the same message. The argument is that when using the randomized hashing scheme H_r instead of the deterministic H, if the attacker can use a different $r' \neq r$ then it has a different signature (since r is part of the signature), and if it uses the same r then it is back in the standard TCR game.

We would like to use the above argument to prove that if the original scheme with the deterministic hash function H is strongly existentially unforgeable then so is the modified scheme with H_r. However, because of some technicality (which is explained later) we cannot prove this for the usual security notion that refers to adaptive chosen-message attacks (CMA). Instead, we describe here two variants of this attack, one where the attacker is weaker than in the CMA game and another in which the attacker is stronger than in the standard CMA game. For these two notions we can prove that indeed using H_r instead of H preserves strong existential unforgeability.

In the first notion, that we call *random-message attack* (RMA), the attacker cannot choose the messages that it wants to see signed, but can only ask for signatures on random messages. In the second notion, that we call super-adaptive chosen-message attack (sCMA) the attacker can first see the r component of the signature and then choose the message to be signed.

RMA Game: The attacker is given the public verification key pk. It can query a signing oracle, each time providing a length 1^ℓ and getting back a triple (M, r, s) where M is a random ℓ-block message and (r, s) a signature on M (as produced by the signature algorithm with the corresponding secret signature key sk).

sCMA Game: The attacker is given the public verification key pk. It can query a signing oracle, each time first obtaining an element r, then specifying a message M and then getting the second element s such that (r, s) is a signature on M (as produced by the signature algorithm with the corresponding secret signature key sk).

In either game, the attacker wins if it outputs a triple (M^*, r^*, s^*) that is different from all the triples that it got from the signing oracle, such that (r^*, s^*) is a valid signature on M^* with respect to the public verification key pk. A signature scheme is called strongly-existentially-unforgeable under RMA (resp. sCMA) is feasible attackers have only insignificant probability of winning the first game (resp. second game).

Below we denote by $SIG[f, g, H]$ the original scheme with the deterministic H, and the new scheme with the randomized H is denoted $SIG[f, g, H_r]$. (Recall that the only difference between these schemes is that the original scheme we compute $\eta = H(M)$ whereas in the new scheme we compute $\eta = H_r(M)$ where $r = f(\rho)$.) Then we have the following theorem:

Theorem 10

1. *If $SIG[f, g, H]$ is strongly existentially unforgeable under random message attack then so is $SIG[f, g, H_r]$.*
2. *If $SIG[f, g, H]$ is strongly existentially unforgeable under super-adaptive chosen message attack then so is $SIG[f, g, H_r]$.*

Proof: We only prove part 2 here. The proof for part 1 is very similar. Given a sCMA attacker A_r for the randomized scheme $SIG[f, g, H_r]$, we construct an sCMA attacker A_d for the "deterministic scheme" $SIG[f, g, H]$.

A_d gets a public key pk as input and forward the same pk to A_r. When A_r makes the first part of its i'th query to the signing oracle then so does A_d, and then A_d forward to A_r the element r_i that it got from the signing oracle. When A_r replies with some message $M_i = (m_{i,1}, \ldots, m_{i,L_i})$ then A_d sets $M'_i = (m_{i,1} \oplus r, \ldots, m_{i,L_i} \oplus r)$ and asks for a signature on M'_i. Then A_d gets the second element s_i and forwards it to A_r. If A_r outputs a forgery (M^*, s^*, r^*) with $M^* = (m^*_1, \ldots, m^*_{L^*})$ then similarly to above A_d outputs a forgery (M', s^*, r^*) with $M' = (m^*_1 \oplus r^*, \ldots, m^*_{L^*} \oplus r^*)$.

Analyzing A_d, it is clear that in all the queries to the signature oracle we have $\eta_i = H(M_i) = H_{r_i}(M'_i) = \eta'_i$ so the answer that A_d returns to A_r is indeed a valid signature of the "deterministic" $SIG[f, g, H]$. Similarly, $\eta^* = H(M^*) = H_{r^*}(M') = \eta'$, so if (r^*, s^*) is a valid signature on M with respect to pk and the "deterministic" $SIG[f, g, H]$ then also (r^*, s^*) is a valid signature on M' with respect to pk and the randomized $SIG[f, g, H_r]$.

It is left to show that if (M^*, s^*, r^*) is different from all the triples (M_i, r_i, s_i) then also (M', r^*, s^*) is different from all the triples (M'_i, r_i, s_i). This is obvious for the case that (r^*, s^*) is different from all the (r_i, s_i)'s. For the case that there exists i such that $r^* = r_i$ and $s^* = s_i$, we know that $M_i = (m_{i,1}, \ldots, m_{i,L_i}) \neq (m^*_1, \ldots, m^*_{L^*}) = M^*$, and since $r^* = r_i$ it follows that also $M'_i = (m_{i,1} \oplus r_i, \ldots, m_{i,L_i} \oplus r_i) \neq (m^*_1 \oplus r^*, \ldots, m^*_{L^*} \oplus r^*) = M'$. □

Comment. Looking at the proof of Theorem 10, it is clear why this argument fails for the case of standard adaptive chosen-message attack: When A_r makes a query M_i to its signing oracle, then A_d must somehow use its own signing oracle to answer that query. A_d would like to query its oracle on M' such that $H(M) = H_{r_i}(M')$, but it only gets r_i from the signing oracle after it commits to M_i, so it can never "hit the right M_i".[8]

[8]Maybe this technicality can be overcome in some variant of the random-oracle model by having A_d expose to A_r a different random oracle than is externally available to it.

D RMX: A Randomization Scheme for Hashing and Signatures

Here we specify RMX, a concrete message randomization scheme that instantiates our $\tilde{H}_r$ scheme from Section 4 and is ready for implementation and standardization. In particular, RMX can serve as a front-end to any hash-then-sign signature scheme resulting, as shown in this paper, in a signature scheme whose security does not depends on full collision resistance of the underlying hash function. Our specification of RMX is geared towards Merkle-Damgård hash functions but it can be adapted to other iterative hash schemes. We also provide some guidance for the use of this scheme with standard signature schemes. We make this appendix as self-contained as possible for those interested in the specification details but want to skip the general theory supporting the scheme as presented in this paper.

In a nutshell, RMX prepends to the message a random string ("salt") of one block, and then XORs the same random string with every block of the message itself (where a "block" is the blocksize used by the underlying hash function). That is, if $m = (m_1, ..., m_n)$ then $RMX(r, m) = (r, m_1 \oplus r, ..., m_n \oplus r)$. We note that the detailed description of RMX from Section D.1 includes a simple padding rule for the last message block; also, to save bandwidth and randomness, the scheme accomodates salt strings shorter than a full message block. RMX can be implemented either as a simple front-end interface to the iterated hash function (leaving the hash implementation unchanged), or it can be integrated with typical implementations of digest functions that read the message block by block and feed these successive blocks into the compression function.

RMX can be used with any hash-then-sign scheme by replacing the digest $H(m)$ in the original signature scheme with $H(RMX(r, m))$. In this case, the salt r is generated for each signature by the signer and transmitted to the verifier together with the message and signature (there is no need to sign r). The verifier uses the regular verification procedure where the original digest function is applied to $RMX(r, m)$ rather than to m. Note that only the signer needs to generate randomness, the verifier receives it with the message/signature. As shown here, off-line collision search is useless against a signature scheme that uses RMX. Rather, to break the signatures the attacker needs to solve a cryptanalytical problem close to finding second preimages (which is a much much harder task than finding collisions). Importantly, to gain this security advantage the value of r must be unpredictable by the attacker before receiving the signature on a given message, and therefore we recommend lengths between 128 bits to a full-block size (with 160 or 256 bits being reasonable default values).

The use of RMX and its application to signatures do not depend on the way the salt r is transmitted; therefore, different applications may choose different ways to transport r. This is analogous to the use of the IV in CBC encryption: the definition of CBC specifies how to use a block cipher to encrypt any-length message given the value of a random IV but it does not determine a specific way to transmit the IV. The interested reader should consult [7] which discusses implementation issues, in particular pointing out to practical options for the transmission of r that incur in minimal changes to existing applications/implementations (the basic idea is to transport r as a parameter under the algorithm identifier associated with the randomized hash transform). The overall conclusion from the design and implementation work in [7] is that the complexity of implementing and deploying RMX in the context of digital signatures is comparable to the effort needed to upgrade existing systems to use a new deterministic hash function, say SHA256. Moreover, once the mechanisms are in place to deal with such upgrade [4], supporting RMX becomes a relatively simple matter.

D.1 The RMX Spec

Given a message m we apply to it a randomization scheme called *RMX* (pronounced *remix*) that takes as inputs the message m and a random string r and produces an output message m'. Informally, a message m' produced by $RMX(r, m)$ is defined as the concatenation of the string r (say, of the length of a hashing block) followed by the result of XOR-ing each block of the message m with the string r followed by a padding rule for the last block of m' defined specifically for the RMX and described below (this RMX padding rule is designed such that the last block of m' is of length a full block less the minimal number of bits required by the padding rule of the underlying hash function H). Following is a precise definition of RMX.

The RMX message randomization scheme for hash function H. Assume H to be a Merkle-Damgård hash function[9] with block size b (in bits), such that H adds at least $c + 1$ bits of padding and length-encoding to each message. For example, all the currently used iterated hash functions pad the input message to a multiple of b bits by appending a single '1' bit followed by as many zeros as needed (possibly none) and then followed by c bits that encode the bit-length of the input message. Typical parameters are $b = 512$, $c = 64$ for SHA1 and SHA256, and $b = 1024$, $c = 128$ for SHA512. For the specification of RMX below we assume that $b < 2^{16}$.

The function *RMX* accepts as inputs a message m of bit-length at most $2^c - b$ and a string r of length between 128 and b bits and produces an output string m' as follows:

1. It computes three strings r_0, r_1, r_2 from r as follows:
   - r_0 is a b-bit string that is obtained from r by padding it with as many zero-bits as needed.
   - r_1 is a b-bit string that is obtained as r concatenated with itself as many times as needed to cover b bits (with the last repetition of r possibly truncated).[10]
   - r_2 is set to the first $b - c - 8$ bits of r_1.

   (Roughly, r_0 will be prepended to the message, r_1 will be XORed to all the blocks except the last, and r_2 will be XORed to the last block.)
2. Parse the message m into $L - 1$ full b-bit blocks $m_1, \dots, m_{L-1}$ and a last block m_L of length b', $1 \le b' \le b$.
3. Set $m'_0 = r_0$.
4. For $i = 1, \dots, L - 1$ set $m'_i = m_i \oplus r_1$.

[9]The same approach can be adapted to other iterative constructions.

[10]For example, if $b = 512$ and $|r| = 160$ then r_1 consists of the concatenation of three times r and then the first 32 bits of r.

5. Let ℓ be a two-octet (16-bit) string, representing the bit-length b' of m_L in big-endian notation. Namely, if ℓ_0, ℓ_1 are the first and second bytes of ℓ, respectively, and each of these bytes represents a number between 0 and 255, then $b' = \ell_1 \cdot 256 + \ell_0$.
   (a) If $b' \leq b - c - 24$ then set m_L^* as a string of $b - c - 8$ bits, obtained by concatenating m_L with as many zero-bits as needed and the 16-bit string ℓ. Namely, in this case $m_L^* = m_L|0^k|\ell$ where $k = b - b' - 16 - c$.
   Set $m'_L = m_L^* \oplus r_2$.
   (b) If $b' > b - c - 24$ then set m_L^* as the concatenation of m_L and as many zero-bits as needed to get a full b-bit block, and set m_{L+1}^* as a string of $b - c - 8$ bits obtained by concatenating as many zero-bits as needed and the 16-bit string ℓ. Namely, in this case $m_L^* = m_L|0^{b-b'}$ and $m_{L+1}^* = 0^{b-c-24}|\ell$.
   Set $m'_L = m_L^* \oplus r_1$ and $m'_{L+1} = m_{L+1}^* \oplus r_2$.
6. Output the string m' as the concatentation of $m'_0, m'_1, \ldots, m'_L$ in case 5(a), and $m'_0, m'_1, \ldots, m'_L, m'_{L+1}$ in case 5(b).

Implementation. Note that the definition of RMX allows for an implementation that acts as a simple front-end interface to the iterated hash function, or it can be integrated with typical implementations of digest functions that read the message block by block and feed these successive blocks into the compression function. See [7] for more information on implementation.

D.2 The use of RMX in Signature Schemes

The main purpose of our randomized hashing algorithm, and specifically the RMX transform, is for use with digital signatures where RMX may preserve the security of the signatures even in the presence of off-line collision attacks.

To compute a signature on a message m using the RMX transform with hash function H (e.g., SHA1, SHA2) and signature algorithm SIG (e.g., RSA or DSA) one proceeds as follows:

1. Choose a random value r as the salt for the RMX transform.
2. Using r and m compute a new message m' following the RMX message randomization scheme defined in Section D.1.
3. Apply H to m'
4. Sign using algorithm SIG the value $H(m')$ to obtain a signature s.
5. Transmit the salt r, message m and signature s to the receiving side.

Note: Steps 2 and 3 are block-wise computations and can be interleaved (or pipelined). That is, there is no need to wait for the full message m' to be computed out of r and m before starting the H computation. In a typical implementation one feeds each block of m into the RMX computation and then feeds the resultant block of m' into the hash function H.

The verification procedure is defined similarly to the above: it receives the three elements r, m, s, computes $m' = \mathit{RMX}(r, m)$ and provides the (randomized) message m' and signature s to the verification procedure (as before the RMX and hash computations can be pipelined).

Note that the above procedure can be used with *any* signature scheme that follows the hash-then-sign paradigm including the two major signature standards: RSA (both deterministic and PSS encoding) [22] and DSS [11].

To support RMX-enabled signatures as above, an application needs to satisfy two requirements: (1) the ability of the signer (not the verifier) to generate the random (unpredictable) salt r; and (2) the ability of the application to accomodate the transmission of r.

We believe that most applications meet these requirements, and even more so given the increasing capabilities of computing devices. In particular, most cryptographic applications already require the ability to generate (pseudo) random bits for key generation, IV's, nonces, or probabilistic signatures such as DSS. As for (2), a great majority of applications can afford the sending of a few extra bytes of salt in addition to a message and signature. See [7] for an in-depth discussion and implementation of a simple mechanism for the transmission of the salt r that works for many different applications. A few more comments are in order here:

1. Observe that the receiver of the signature can only start to hash the message after it knows the salt. Hence, in applications where buffering the entire incoming message is impractical, it is preferable to send the salt *before* the message. In particular, in such applications one probably cannot make the salt be a component of the signature itself (since typically a signature is transmitted after the message). Also, we stress that an application using RMX must ensure that an attacker cannot choose the message to be signed (or part of its contents) after seeing the salt. Hence the salt, even if sent before the message, will be sent to the verifier only after the message has been fully determined.

2. For extremely bandwidth-limited applications, one can sometime save on bandwidth by including the salt in the signature (even if it means sending the salt after the message). For example, with DSS signatures one can re-use the random component $r = g^k$ that already exists in the signature also as the hashing salt, thus preserving the original data size. It should be noted, however, that this means that the quantity $r = g^k$ must be computed before computing the message digest. In the case of RSA-PSS [22] an approach similar to DSS can be used to save bandwidth (here the randomness used internally by the signature can be recovered by the recipient of the signature via the RSA-verification operation). The situation is more problematic with the deterministic RSA encoding of PKCS#1 v1.5 [22]; here the only way to preserve bandwidth is to include the salt r under the signature itself. That is, instead of applying the RSA operation solely to the result of the randomized hash operation one applies it to the concatenation of this result and the salt r. In this way, the recipeient of the signature can recover the salt via the RSA-verification procedure. This, however, requires a change in the message encoding of PKCS#1 v1.5, and hence less desirable as a general solution.

3. We end by noting that using an independent salt value has the additional advantage that it allows for the pre-computation of the randomized hash value (i.e., one can choose r, compute $d = H(RMX(r, m))$ and store the triple (r, d, m), such that upon a request for a signature on m one computes the signature directly on the pre-computed d). Also, it supports multi-level hashing which is essential in some cases, for example the XML signatures treated in [7].