J. Cryptol. (2012) 25: 748–779
DOI: 10.1007/s00145-011-9109-y

Journal of
CRYPTOLOGY

Security Analysis of Randomize-Hash-then-Sign Digital Signatures*

Praveen Gauravaram and Lars R. Knudsen

Department of Mathematics, Technical University of Denmark, Matematiktorvet, Building S303, 2800, Kgs. Lyngby, Denmark
P.Gauravaram@gmail.com; lars@ramkilde.com

Communicated by Antoine Joux

Received 8 October 2009
Online publication 12 October 2011

Abstract. At CRYPTO 2006, Halevi and Krawczyk proposed two randomized hash function modes and analyzed the security of digital signature algorithms based on these constructions. They showed that the security of signature schemes based on the two randomized hash function modes relies on properties similar to the second preimage resistance rather than on the collision resistance property of the hash functions. One of the randomized hash function modes was named the RMX hash function mode and was recommended for practical purposes. The National Institute of Standards and Technology (NIST), USA standardized a variant of the RMX hash function mode and published this standard in the Special Publication (SP) 800-106.

In this article, we first discuss a generic online birthday existential forgery attack of Dang and Perlner on the RMX-hash-then-sign schemes. We show that a variant of this attack can be applied to forge the other randomize-hash-then-sign schemes. We point out practical limitations of the generic forgery attack on the RMX-hash-then-sign schemes. We then show that these limitations can be overcome for the RMX-hash-then-sign schemes if it is easy to find fixed points for the underlying compression functions, such as for the Davies-Meyer construction used in the popular hash functions such as MD5 designed by Rivest and the SHA family of hash functions designed by the National Security Agency (NSA), USA and published by NIST in the Federal Information Processing Standards (FIPS). We show an online birthday forgery attack on this class of signatures by using a variant of Dean's method of finding fixed point *expandable messages* for hash functions based on the Davies-Meyer construction. This forgery attack is also applicable to signature schemes based on the variant of RMX standardized by NIST in SP 800-106. We discuss some important applications of our attacks and discuss their applicability on signature schemes based on hash functions with 'built-in' randomization. Finally, we compare our attacks on randomize-hash-then-sign schemes with the generic forgery attacks on the standard hash-based message authentication code (HMAC).

Key words. Collision resistance, Compression function, Davies–Meyer, Digital signature, Hash function, Merkle–Damgård, Randomized hashing, RMX, Second preimage resistance, SHA-3 hash function competition.

* This paper was solicited from EUROCRYPT 2009.

1. Introduction

1.1. *Hash Functions and Digital Signatures*

A cryptographic hash function is a tool which processes a message of arbitrary length to a hash value of fixed length. The length of the message and its hash value are measured in the number of bits or bytes. In general, a hash function with t-bit hash value is called a t-bit hash function. The fundamental security properties of a hash function are collision resistance, second preimage resistance and preimage resistance [54]. A hash function is *collision resistant* if it is computationally infeasible to find two distinct messages that have the same hash value. A hash function is *second preimage resistant* if, for a given target message, it is computationally infeasible to find a message distinct from the target message such that the hash values of these two messages are the same. A hash function is *preimage resistant* if, for a given target hash value, it is computationally infeasible to find a message which hashes to the given target hash value. The term 'computational infeasibility' means that the complexity of an algorithm to break any of these properties is not less than that of the generic attack required to break that property. The generic attack to break the collision resistance property of a t-bit hash function is called a birthday collision attack, and it requires $2^{t/2}$ (distinct) queries to the hash function. The generic attack to break the second preimage resistance and preimage resistance properties of a t-bit hash function requires on average 2^t (distinct) queries to the hash function. Formal definitions for these security properties were given by Rogaway and Shrimpton [73].

Efficient digital signature generation and verification is one of the main applications of hash functions. In this application, a signer uses a hash function H to compress a message m, that he or she intends to sign, to a fixed size hash value $H(m)$. The signer then signs the hash value using the signature generation algorithm SIG to produce the signature $s = \mathsf{SIG}(H(m))$. The signer sends (m, s) to the intended receiver, who verifies the signature s on m by using the verification algorithm VER which outputs a bit 1 upon successful verification. Digital signature algorithms that use hash functions in this way are called hash-then-sign digital signature algorithms. When the underlying hash function H is not *collision resistant*, a signature scheme SIG can be forged by asking a signer to sign a legitimate message m and then showing $\mathsf{SIG}(H(m))$ as the signature of a fraudulent message n where $m \neq n$ and $H(m) = H(n)$. Hence, a hash function used in a hash-then-sign scheme must be *collision resistant* for the security of the scheme against forgery attacks.

Several hash functions were designed exclusively for the digital signature application. Some notable proposals are MD4 [71] and MD5 [72] by Rivest and the SHA family of hash functions [60,63–65] by the National Security Agency (NSA), USA and published by the USA's government standards agency, the National Institute of Standards and Technology (NIST). The SHA family includes SHA-0 published in the Federal Information Processing Standard (FIPS 180) [63] and its successors SHA-1, SHA-224, SHA-256, SHA-384 and SHA-512 that were published in the standard FIPS 180-3 [60].[1] All these designs are based on the Merkle–Damgård [20,55] iterated hash function design

[1] The standard FIPS 180-3 was preceded by the standards FIPS 180-2 [65], which has SHA-1, SHA-256 and SHA-512, and FIPS 180-1 [64] in which SHA-1 was first published.

paradigm with the exception that SHA-224 and SHA-384 truncate their final 256-bit and 512-bit hash values to 224 and 384 bits respectively.

The MD4, MD5, SHA-0 and SHA-1 hash functions are no longer collision resistant, and several collision attacks on their reduced versions [12,26,27], compression functions [18,29] and full hash functions [12,28,31,52,83,84] were demonstrated. Of these hash functions, MD5 and SHA-1 are widely implemented in many information processing applications. Nearly a decade before collisions were found for the MD5 hash function [85], Dobbertin [30] suggested not to implement MD5 as a collision resistant hash function after his pseudo collision attack on MD5 [30]. Although no practical collisions have been found for SHA-1, a theoretical collision attack on SHA-1 by Wang, Yin and Yu [84] in 2005 demonstrated that SHA-1 is not collision resistant. Soon after the formal notice of this attack, NIST announced that federal agencies must stop relying on digital signatures that are generated using SHA-1 by the end of 2010. Considering the wide usage of MD5 and SHA-1 in the signature algorithms such as RSA [45,74] and DSA [61,62], there has been a growing amount of research showing how seriously these attacks (in particular on MD5) could undermine the security of digital signatures [10, 38,49,79,80] in which these hash functions are deployed.

1.2. *Randomize-Hash-then-Sign Digital Signatures*

For a long time, it has been suggested to randomize each message using a fresh random value, also called a salt, before it is hashed and signed so that the security of the digital signatures depends on a property *weaker* than the collision resistance of the hash function [1,23,24,56]. Message randomization before hashing would require an attacker to predict the random value in order to make use of collisions in a hash function to forge a digital signature. The composition functions of message randomization transforms and hash functions are called randomized hash functions.

Shortly after the discovery of collision attacks on several popular hash functions by Wang et al. [81,82,84,85], Halevi and Krawczyk [40] proposed two concrete randomized hash function constructions to strengthen the security of digital signature schemes against collision attacks on hash functions. These randomized hash function constructions use Merkle–Damgård hash function construction as a black-box and were intended for use with standard digital signature schemes without the need to make any changes to either the underlying hash functions or the signature algorithms. When a message is processed using the first randomized hash function of [40], the signature must be generated on both the hash value and the random value used to randomize the message. However, certain standard digital signature algorithms such as DSA and RSA do not support signing both the random value and the hash value. To overcome this practical constraint, Halevi and Krawczyk proposed another randomized hash function called RMX [40, Appendix D], [41] which does not require the hash values generated under it to be signed together with the random value. They also suggested RMX for adoption into practice. An implementation of RSA based on RMX-SHA-1 was discussed in [43]. NIST standardized a variant of RMX in the special publication (SP) 800-106 [21]. In addition, it is welcomed by NIST, that the candidate hash functions in the SHA-3 hash function competition support randomized hashing [66].

In an RMX-hash-then-sign signature scheme, the signer computes the signature of a message m as follows: He chooses a random value denoted r, and randomizes m by

passing the pair (r, m) as input to the RMX transform. The randomized message is given by $M = \text{RMX}(r, m)$. The signer processes the message M using a t-bit hash function H and obtains the t-bit hash value $H(M)$. The signer signs the hash value $H(M)$ using a signature algorithm, denoted SIG, and obtains the signature s. The signer sends the triplet (m, r, s) to the verifier, who computes $M = \text{RMX}(r, m)$ and provides the pair (M, s) to the verification procedure VER to verify s. In this article, signature schemes based on the two randomized hash functions (including the RMX hash function) proposed in [40] are called randomize-hash-then-sign signature schemes and those based on the RMX hash function are called RMX-hash-then-sign signature schemes.

1.3. *Analytical Results of Halevi and Krawczyk*

As noted before, security of the randomized hash functions would depend on properties *weaker* than the collision resistance property of hash functions. For the first randomized hash function proposed by Halevi and Krawczyk [40], this property was called target collision resistance (TCR) and for the RMX hash function, the property was referred to as enhanced target collision resistance (eTCR). Halevi and Krawczyk proved that randomized hash functions are TCR and eTCR if their underlying compression functions possess two security properties called chosen second preimage resistance and evaluated second preimage resistance. Both these properties are related to the second preimage resistance property of the compression function.

Finally, the analysis of [40] argues that an attacker who can break the TCR property of the first randomized hash function and the eTCR property of the RMX hash function can forge signature schemes based on these hash functions. It was also claimed that to break randomize-hash-then-sign signature schemes an attacker must solve a cryptanalytical problem close to finding second preimages, which is a much harder task than finding collisions in a hash function. They also observed that the generic second preimage attack of Kelsey and Schneier [46] on the Merkle–Damgård hash functions is applicable to their two randomized hash functions. This attack breaks the TCR and eTCR properties of the two randomized hash functions and has complexity far beyond the bound of a birthday attack.

Definitions of two randomized hash functions and their respective TCR and eTCR properties as well as chosen and evaluated second preimage resistance properties are provided in Sect. 3.

1.4. *Related Work*

Dang and Perlner [22] showed a generic chosen message forgery attack on signature schemes based on a t-bit RMX hash function in $2^{t/2}$ chosen messages and $2^{t/2}$ operations of the hash function and a similar amount of memory. This attack produces a collision of the form $H(\text{RMX}(r, m)) = H(\text{RMX}(r^*, n))$ which implies $\text{SIG}(m) = \text{SIG}(n)$, where m is one of the chosen messages, n is the message corresponding to the forged signature and $(r, m) \neq (r^*, n)$.

1.5. *Our Results*

In this article, we first note that the attack of Dang and Perlner [22] does not produce a verifiable forgery if the signer uses the same random value for both RMX hashing and

signing for randomized signature algorithms such as DSA [61,62], ECDSA [4,62] and RSA-PSS [45,74]. The idea of re-using the random value present in signature schemes for the randomized hashing to save the communication bandwidth was addressed in [21, 40,57,67]. The attack of Dang and Perlner also does not apply to signature schemes based on the other randomized hash function analyzed by Halevi and Krawczyk wherein both the random value and hash value are signed.

We then show a generic chosen message existential forgery attack [39] on the RMX-hash-then-sign signature schemes when the compression function of the hash function has fixed points that are easy to find. Our attack produces a valid forgery in the applications where the forgery attack of Dang and Perlner does not succeed. In our attack we use a subtle variant of Dean's method of finding fixed point expandable messages [25, 46] for hash functions based on the compression functions vulnerable to easily found fixed points.[2] Many popular hash functions, which include MD4 [71], MD5 [72], the SHA family [65] and Tiger [2], have compression functions that use the Davies–Meyer construction [44,68] for which fixed points can easily be found [58]. In an existential forgery attack, the attacker asks the signer for the signatures on a set of messages of his choice and is then able to produce a valid signature on a message which was never signed by the signer. Our forgery attack requires $2^{t/2}$ equal length chosen messages, $2^{t/2+1}$ offline operations of the compression function and has a probability of 2^{-24} to hit the correct bits used to pad the message according to the RMX transform. The attack requires about $2^{t/2}$ words of memory. With this computational complexity, we can establish a collision of the form $H(\text{RMX}(r, m)) = H(\text{RMX}(r, m\|n))$, where m is one of the chosen messages and $n \neq m$ is a randomized message block which gives a fixed point. This implies $\mathsf{SIG}(m) = \mathsf{SIG}(m\|n) = s$ and therefore $(m\|n,s)$ is the forgery of (m,s). This attack demonstrates that the RMX hash function instantiated with the Davies–Meyer construction gains implementation benefits at the expense of online birthday forgery attacks on the RMX-hash-then-sign signature schemes.

Variants of our forgery attacks as well as those of [22] on the randomize-hash-then-sign signature schemes require less than $2^{t/2}$ queries to the signer and less than $2^{t/2}$ memory at the expense of more than $2^{t/2+1}$ (but still less than 2^t) offline compression function computations. Our attacks also apply to signature schemes that use a variant of RMX published by NIST in SP 800-106 [21]. They also apply to signature schemes based on the randomized hashing for some of the variants of the Merkle–Damgård construction, such as HAIFA [13] and the construction with split padding [87].

Finally, to develop an understanding of the differences between the generic attacks on randomized hash functions and those applied on the standard keyed hash functions such as hash-based message authentication code (HMAC), we compare the generic attack models used to forge randomize-hash-then-sign signature schemes presented in this article to those applied on HMAC [7,8].

1.6. *Comparison of Our Analysis with that of Halevi and Krawczyk*

In [40] it was formally shown that for the randomized hash functions to be TCR and eTCR, it is sufficient for the compression functions to have chosen second preimage

[2] Fixed point expandable messages were used to mount long message second preimage attacks in the Merkle–Damgård hash function [25,46] and some of its variants [3,34,35].

resistance and evaluated second preimage resistance properties. As noted before, both these properties are related to the second preimage resistance of the compression function. In other words, the analysis of Halevi and Krawczyk shows a relation between second preimage resistance of the compression function of a given hash function and the difficulty of breaking TCR and eTCR properties in the corresponding randomized hashing schemes. Breaking these properties of randomized hash functions is much harder than finding collisions for a hash function. An attacker who breaks the TCR (resp. eTCR) property of the first randomized hashing scheme (resp. RMX) can then forge the digital signature based on this hash function by making one query to the signature algorithm. We remark that the analysis of [40] does not provide any explicit security bounds required to break the TCR and eTCR properties of two randomized hash functions when their compression functions are chosen second preimage resistant and evaluated second preimage resistant. It also does not provide any explicit security bounds required to forge randomize-hash-then-sign signature schemes when the underlying randomized hash functions are TCR and eTCR. Since it was claimed that to break randomize-hash-then-sign signature schemes an attacker must solve a cryptanalytical problem close to finding second preimages, one may observe that the complexity of breaking TCR and eTCR properties of a t-bit randomized hash function is close to 2^t hash function computations, which is a much harder task than finding collisions in a hash function by a birthday attack in about $2^{t/2}$ complexity. The analysis of [40] also observes that the application of a Kelsey–Schneier [46] second preimage attack on t-bit randomized hash functions for a target message of length 2^d blocks leads to an *existential* forgery attack on the corresponding randomize-hash-then-sign signature scheme in 2^{t-d} offline operations of the compression function and one query to the signer of a randomize-hash-then-sign signature algorithm.

Our analysis aims to present an upper bound of security for the randomize-hash-then-sign schemes against the adversaries who make several queries over online to the signer. As shown by our attacks, an attacker who can make about $2^{t/2}$ queries to the signer of a randomize-hash-then-sign signature algorithm, can mount *existential* forgery attacks on the randomize-hash-then-sign signatures with a similar amount of offline computations and memory, where t is the size of the hash value. This result illustrates the exact security of the randomize-hash-then-sign schemes with respect to online attacks and shows that the security of these schemes does not necessarily rely on the properties similar to the second preimage resistance of the hash functions, as better attacks can be found in an online model. Our analysis also shows that compression functions with easily found fixed points, such as the Davies–Meyer construction, can be exploited in the RMX hash function setting to forge digital signature algorithms that can use the same random value for both RMX hashing and signing, as noted in Sects. 1.5 and 4.1.1.

In summary, our analysis of randomize-hash-then-sign schemes follows an attack-oriented approach in comparison to the proof-oriented approach pursued by Halevi and Krawczyk. In addition, we analyzed the security of these schemes against adversaries who make several queries with the same or different messages to the signer, whereas Halevi and Krawczyk analyzed the security of randomized hash function modes against an adversary who makes only one message query to the signer. Both these analytical results show that it is not possible to forge randomize-hash-then-sign signatures by employing only offline birthday collision attacks on the hash functions. As shown by our

analysis, collision attacks must be online and with about birthday attack complexity one can forge randomize-hash-then-sign algorithms. Thus, our analysis complements the analysis of Halevi and Krawczyk by showing that randomized hash functions achieve an essential security improvement over the *offline* birthday collision attacks by forcing these attacks to work *online* (e.g. requiring $2^{t/2}$ messages signed by the legitimate signer and similar amount of memory).

1.7. *Impact of Our Results*

Our generic forgery attacks on the randomize-hash-then-sign signature schemes using fixed points in the compression functions are totally impractical for a typical hash value size of 256 bits. Moreover, our attacks cannot be parallelized, as they require a real signer to sign a very large set of messages. Our analytical results are in no contradiction to those of Halevi and Krawczyk [40] for the reasons stated in Sect. 1.6. Although variants of our forgery attacks require less than $2^{t/2}$ legitimate signatures and memory, they require more than $2^{t/2+1}$ (but less than 2^t) offline compression function computations. The forgery attack of [22] on the RMX-hash-then-sign signature schemes has a similar complexity, though its application is more limited.

1.8. *Guide to the Article*

In Sect. 2, we define the notation and fundamentals of hash functions and digital signatures. In Sect. 3, we define randomized hash functions of [40] and the variant of RMX hash function mode standardized by NIST [21]. In Sect. 4, we present the forgery attack of [22] on the RMX-hash-then-sign schemes and discuss its limitations. In Sect. 5, we show how to apply Dean's method of building fixed point expandable messages to forge hash-then-sign signatures. In Sect. 6, we argue that offline birthday collision attacks are not useful to forge randomize-hash-then-sign digital signatures. In Sect. 7, we describe our forgery attack on the RMX-hash-then-sign signature algorithms and discuss variants of this attack. In Sect. 8, we present some important applications of our forgery attacks. In Sect. 9, we discuss the applicability of our analysis to signature schemes based on hash functions with a 'built-in' randomization feature. In Sect. 10, we compare the generic attack models on randomize-hash-then-sign schemes to those on HMAC. Section 11 holds the conclusion.

2. Preliminaries

2.1. *Notation*

For a $t \in \mathbb{Z}^+$, $\{0,1\}^t$ defines the set of all bit strings of length t. For two bit strings a and b, $a\|b$ defines the concatenation of a and b. For example, if $a = 000$ and $b = 11$ then $a\|b = 000\|11 = 00011$. For a bit string a, the left most bit of a is its most significant bit (MSB) and the right most bit of a is its least significant bit (LSB). For $\alpha \in \mathbb{N}$, the first α bits of a are called the first α MSBs of a and the last α bits of a are called the last α LSBs. For example, for $a = 10100010$, the bit 1 is the MSB and the bit 0 is the LSB and the bits 1010 are the first four MSBs of a and the bits 0010 are the last four LSBs. For $\alpha \in \mathbb{N}$ and $e \in \{0,1\}$, e^α defines the concatenation of e bit for α times. For example,

$1^4 = 1\|1\|1\|1 = 1111$. For a bit string a and $\alpha \in \mathbb{N}$, $a^{[\alpha]}$ defines consecutive α bits of a starting from its MSB. For example, if $a = 1011011001$ then $a^{[4]} = 1011$. Similarly, for $a^b \in \mathbb{Z}^+$, $(a^b)^{[\alpha]}$ defines consecutive α bits of a^b starting from its MSB. The number of bits in a bit string a is denoted by $|a|$. For example, if $a = 0^{256}\|1^{128}$ then $|a| = 384$.

2.2. *Merkle–Damgård Hash Function Construction*

The Merkle–Damgård construction [20,55] is a popular hash function mode of operation which has been used in the design of popular hash functions such as MD5, the SHA family and the Whirlpool hash functions. A compression function which takes a fixed input length value and outputs a fixed length hash value is the core component of this construction. Let $h : \{0, 1\}^b \times \{0, 1\}^t \to \{0, 1\}^t$ be a compression function which takes a b-bit message block and a t-bit chaining value as inputs and produces a t-bit output chaining value. Let $H : \{0, 1\}^* \to \{0, 1\}^t$ be the Merkle–Damgård construction built by iterating the compression function h in order to process a message of arbitrary length. Often, an upper bound on the length of the message which can be hashed is specified for H. Let this length be 2^l bits. A message m to be processed using H is always padded in a manner such that the length of the padded message is a multiple of the block length b of h. Let $|m|$ be the l-bit binary representation of the length of the message m. The message m is padded as follows:

Divide m into b-bit blocks m_i for $i = 1$ to L and let q be the number of message bits in the last block m_L where $q \leq b$.

- If $q \leq b - l - 1$ then the message m is padded with $1\|0^{b-l-q-1}\||m|$.
- If $q > b - l - 1$ then the message m is padded with $1\|0^{b-q-1}$ and an additional b-bit block $0^{b-l}\|\ |m|$ is concatenated to the padded message $m\|1\|0^{b-q-1}$.

Every message block m_i is processed by iterating h as defined by the expression $H_i = h(H_{i-1}, m_i)$, where H_0 is the initial value (IV) of H, H_i is the intermediate chaining value of H at the i^{th} iteration of h and H_L is the hash value of H. In this article, the Merkle–Damgård hash function with H_0 as the IV is denoted by H^{H_0}.

2.3. *Fundamental Security Properties*

Some fundamental security properties of an iterated hash function H^{H_0} (e.g. Merkle–Damgård) instantiated with an ideal compression function h and an ideal compression function h that are essential for the analysis presented in this article are listed below.

2.3.1. *Properties of an Iterated Hash Function* H^{H_0}

1. Collision resistance (CR): It should take $2^{t/2}$ operations of H^{H_0} to find two messages m and n such that $m \neq n$ and $H^{H_0}(m) = H^{H_0}(n)$.
2. Second preimage resistance (SPR): For a challenged target message m and its hash value under H^{H_0}, it should take 2^t operations of H^{H_0} to find another message n such that $n \neq m$ and $H^{H_0}(m) = H^{H_0}(n)$. We recall that for a target message of 2^d blocks, a second preimage for a Merkle–Damgård hash function can be found in 2^{t-d} operations of the compression function [46].

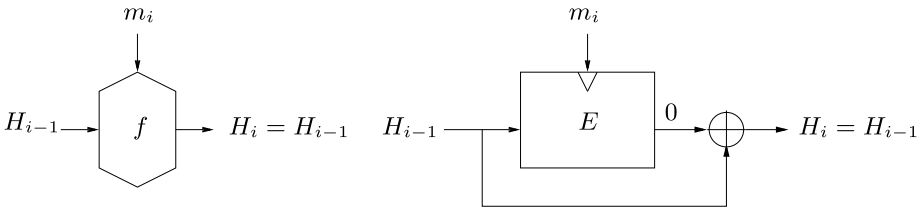

Fig. 1. Illustration of fixed point attack for the Davies–Meyer compression function.

2.3.2. *Properties of an Ideal Compression Function h*

1. Collision resistance: It should take $2^{t/2}$ operations of h to find two different pairs (H_{i-1}, m_i) and (H^*_{i-1}, n_i) such that $h(H_{i-1}, m_i) = h(H^*_{i-1}, n_i)$.
2. Second preimage resistance: For a challenged pair (H_{i-1}, m_i), it should take 2^t operations of h to find a pair (H^*_{i-1}, n_i) such that $(H_{i-1}, m_i) \neq (H^*_{i-1}, n_i)$ and $h(H_{i-1}, m_i) = h(H^*_{i-1}, n_i)$. This property is also called random-SPR (r-SPR) [40].

2.4. *Compression Functions with Fixed Points*

The Davies–Meyer compression function construction is one of the twelve single block length block cipher-based compression functions that were proved to be collision resistant and (second) preimage resistant in the ideal cipher model [17,68]. This construction has been used in the design of many popular hash functions such as MD4, MD5 and the SHA family and recently in SHAvite-3 [14], a second round candidate of the NIST's SHA-3 hash function competition. Let h be the Davies–Meyer compression function; then h is defined by $h(H_{i-1}, m_i) = E_{m_i}(H_{i-1}) \oplus H_{i-1} = H_i$, where m_i is the message block which is used as a key to the block cipher E, the input chaining value H_{i-1} is the plaintext to E and $\oplus$ is the feed-forward operation.

A fixed point for a compression function h is a pair (H_{i-1}, m_i) such that $h(H_{i-1}, m_i) = H_{i-1}$. A fixed point for h can be easily found by evaluating the expression $E^{-1}_{m_i}(0)$ for some message block m_i [58], [46, Appendix A.1] as illustrated in Fig. 1. Note that, for every unique message block m_i, there exists one and only one fixed point for the Davies–Meyer compression function. Fixed points can also be found for a variant of this compression function with addition modulo 2^t as the feed-forward operation. Similarly, it is easy to find fixed points for three other provably collision resistant and second preimage resistant single block length compression functions [17,68].

2.5. *Existential Forgery Attack on the Digital Signature Schemes*

In this article, we consider an adversary who tries to make an existential forgery attack on a digital signature scheme SIG under a generic *adaptive chosen message attack* model [39]. In this attack, an attacker first queries the signer (also called the challenger) *adaptively* for signatures on several chosen messages. The attacker then outputs a new message which was not queried to the signer and its signature as the forgery. The attack is *adaptive*, as the adversary can choose queries that depend on both the challenger's public key and signatures obtained for the previous queries. This attack is regarded as

the most severe natural attack an adversary can mount on the signature scheme [39]. The sequence of steps associated with this attack are outlined below:

1. The challenger generates a public and a private key pair (Pk, Sk) using a key generation algorithm. An attacker receives the public key Pk.
2. The attacker queries a list of q messages $m^1, \ldots, m^q$ to the challenger, and the attacker can choose these queries *adaptively*.
3. The challenger generates the signatures s_i for the messages m^i by using his private key Sk and the signature algorithm SIG, where $i = 1, \ldots, q$. The challenger sends the signatures s_i to the attacker.
4. The attacker is considered to have existentially forged the signature scheme SIG by returning a pair (m, s) satisfying:
   (a) $(m, s) \notin \{(m^1, s_1), \ldots, (m^q, s_q)\}$; and
   (b) The triplet (Pk, m, s) produces a valid verification. That is, the verification algorithm $\mathsf{VER}(Pk, H(m), s)$ outputs a bit 1.

Let Adv be the probability that the adversary succeeds in this attack, taken over the coin tosses made by it and the challenger. The adversary is said to (t, q, ϵ)-existentially forge the signature scheme SIG if it runs in at most t time, makes at most q queries and $\mathsf{Adv} \geq \epsilon$.

3. Randomized Hashing

The notion of universal one-way hash functions (UOWHFs) was introduced by Naor and Yung [59], who showed how to construct UOWHF primitives based on any 1-1 one-way function and proposed digital signature schemes based on UOWHFs. The UOWHFs were called target collision resistant (TCR) hash functions by Bellare and Rogaway [9], a notion which was also adopted by Halevi and Krawczyk [40]. Bellare and Rogaway [9] and later Shoup [78] proposed and analyzed composition constructions to build TCR hash functions from TCR compression functions.

3.1. *TCR Hash Function Mode*

A family of hash functions $\{H_r\}_{r \in R}$ for some set R is TCR [9,40,59,78] if no efficient attacker can win the following game except with negligible probability:

1. First choose a message m and then receive a salt or random value $r \in_R R$.
2. Find a second message n such that $m \neq n$ and $H_r(m) = H_r(n)$.

A signature scheme based on a TCR hash function is called a TCR-hash-then-sign signature scheme [9,78]. Halevi and Krawczyk [40] proposed a TCR hash function mode H_r for the Merkle–Damgård hash function H^{H_0}. In this scheme, every block m_i of the message m is mixed with a b-bit random value r by using an exclusive-or operation before it is processed with the hash function H^{H_0}. The scheme H_r with H_0 as the IV and a hash value of size t bits is formally defined as follows:

$$H_r^{H_0}(m) = H_r^{H_0}(m_1 \| \ldots \| m_L) \stackrel{\text{def}}{=} H^{H_0}(m_1 \oplus r \| m_2 \oplus r \| \ldots \| m_L \oplus r).$$

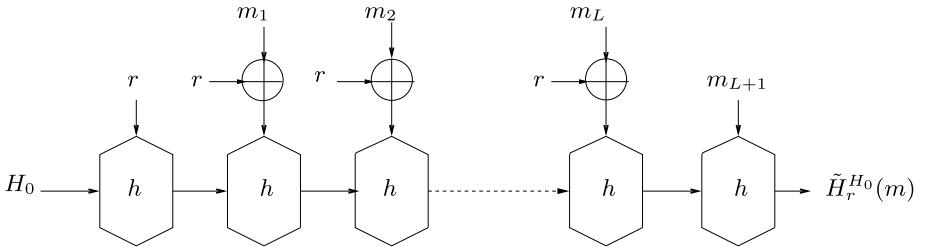

Fig. 2. Illustration of RMX hash function mode.

To sign a message m, a user of the TCR-hash-then-sign scheme first chooses a salt r and then invokes the underlying digital signature scheme to sign the pair $(r, H_r(m))$. However, signing this pair requires modifications to some of the standard digital signature schemes, similar to the schemes proposed in [9,78], which is undesirable in practice as a general solution. For instance:

- Signature algorithms such as DSA [61,62] and ECDSA [4] do not accommodate for the signing of both r and $H_r(m)$ as they are designed to compute signatures only for the hash value $H_r(m)$ and do not have a field to accommodate r.
- Signature schemes such as the traditional deterministic RSA encoding of PKCS#1 v1.5 [45,74] can accommodate both r and $H_r(m)$ under the signed block (since RSA moduli are long enough) [40]. However, the PKCS#1 standard does not accommodate the random value r but only the hash value [40]; to accommodate r, explicit changes to this standard are required, which is possible but difficult [48].

3.2. *eTCR Hash Function Mode*

To solve the problem of signing both r and $H_r(m)$ in the TCR-hash-then-sign schemes, Halevi and Krawczyk introduced an improved notion for the TCR hash function family called the enhanced TCR (eTCR) hash function family [40]. A hash function family $\{\tilde{H}_r\}_{r \in R}$ for some set R is eTCR if there exists no efficient attacker who can win the following game except with negligible probability:

1. First choose a message m and then receive a salt or random value $r \in_R R$.
2. Find a second message n and a salt r^* such that $(r^*, n) \neq (r, m)$ and $\tilde{H}_r(m) = \tilde{H}_{r^*}(n)$.

Halevi and Krawczyk [40] proposed an eTCR hash function mode, denoted $\tilde{H}_r$, as a suitable randomized hash function mode for use in digital signatures, as it does not require explicit signing of the salt r. The eTCR hash function $\tilde{H}_r$ with H_0 as the IV and a hash value of size t bits is formally defined as follows:

$$\tilde{H}_r^{H_0}(m) = \tilde{H}_r^{H_0}(m_1 \| \ldots \| m_L) \stackrel{\text{def}}{=} H^{H_0}(r \| m_1 \oplus r \| m_2 \oplus r \| \ldots \| m_L \oplus r).$$

Figure 2 depicts the eTCR hash function mode $\tilde{H}_r$ assuming that the last message block m_L of m is complete (i.e. the message m is a multiple of b bits). Hence, an

additional padding block m_{L+1} which contains the bits $1\|0^{b-l-1}\||m|$ is appended to m. The designers proposed a concrete specification of $\tilde{H}_r$ called RMX together with its implementation details [40, Appendix D], [41,43]. The major focus of this article is the analysis of signature schemes based on the RMX hash function mode.

3.2.1. *RMX Specification*

The RMX hash function randomizes an input message m of at most $2^l - b$ bits by using a random value r of size between 128 and b bits and outputs a message M. Following [40, Appendix D], [41,43], we define the RMX algorithm as below:

1. Compute three random values r_0, r_1 and r_2 from r as follows:
   (a) $r_0 = r\|0^{b-|r|}$ such that $|r_0| = b$ bits.
   (b) $r_1 = \underbrace{r\|r\|\ldots\|r}_{b \text{ bits}}$ such that $|r_1| = b$ and the last repetition of r is truncated if needed.
   (c) $r_2 = r_1^{[b-l-8]}$ (i.e. the first $b-l-8$ MSBs of r_1 are assigned to r_2).
2. Divide the input message m into equal size b-bit blocks. Assume that the size of m is $L-1 \times b + b'$ bits, where $L-1 \times b$ bits are divided into $L-1$ b-bit blocks $m_1, m_2, \ldots, m_{L-1}$ and the last b' LSBs of m are placed in a block m_L of length b' where $1 \leq b' \leq b$.
3. Assign $M_0 = r_0$.
4. For $i = 1$ to $L-1$:
   (a) $M_i = m_i \oplus r_1$.
5. Let *lpad*, which means the last block pad, be a 16-bit string representing the bit length b' of m_L in big-endian notation. Let $lpad_0$ and $lpad_1$ be the first and second bytes of *lpad* respectively, and each of these bytes represents a number between 0 and 255. This implies that $b' = 256 \times lpad_1 + lpad_0$.
   (a) If $b' \leq b-l-24$ then assign $M_L^* = m_L\|0^k\|lpad$ where $k = b-b'-16-l$. Assign $M_L = M_L^* \oplus r_2$.
   (b) If $b' > b-l-24$ then assign $M_L^* = m_L\|0^{b-b'}$ and $M_{L+1}^* = 0^{b-l-24}\|lpad$. Assign $M_L = M_L^* \oplus r_1$ and $M_{L+1} = M_{L+1}^* \oplus r_2$.
6. Output the randomized message M, where $M = \text{RMX}(r, m) = M_0\|\ldots\|M_L$ in the case of (5) and $M = \text{RMX}(r, m) = M_0\|\ldots\|M_L\|M_{L+1}$ in the case of (5).
7. Process M with H^{H_0} to obtain a t-bit hash value.

Remark 1. We note that when $b' \leq b-l-24$ with $k = b-b'-16-l$, an additional b-bit block must be appended to M_L to accommodate the padding and message length encoding bits of size at least $l+1$ required by the hash function H^{H_0} to process M. We illustrate this observation in Appendix A. In addition, when $b' \leq b-l-24$, $|M_L^*| = b-l$ and hence $|r_2| = b-l$ bits which means $r_2 = r_1^{[b-l]}$. For example, when $|m_L| = b' = b-l-24$ bits, $k = b-b'-l-16 = 8$ bits. Now $M_L^* = m_L\|0^8\|lpad$ and $|M_L^*| = b-l-24+8+16 = b-l$ bits. Hence, r_2 must also be of size $b-l$ bits. Therefore, in this article, we assign $r_2 = r_1^{[b-l]}$ for $b' \leq b-l-24$ bits.

Remark 2. Instead of $k = b-b'-16-l$, if one assigns $k = b-b'-24-l$ for $b' \leq b-l-24$ bits, then the padding and message length encoding bits required by

H^{H_0} to process M can be accommodated in the last block of M itself, as illustrated in Appendix A. When $k = b - b' - 24 - l$ with $b' \leq b - l - 24$, $|M_L^*| = b - l - 8$ bits and hence $|r_2| = b - l - 8$ bits, which means $r_2 = r_1^{[b-l-8]}$ which is the same as in the RMX specification. For example, let $|m_L| = b' = b - l - 24$ bits. Then $k = 0$ bits. Now $M_L^* = m_L \| 0^0 \| lpad$ and $|M_L^*| = b - l - 24 + 0 + 16 = b - l - 8$ bits; hence $|r_2| = b - l - 8$ and one must have assigned $r_2 = r_1^{[b-l-8]}$.

3.2.2. *Standardization of RMX*

NIST standardized a variant of RMX in SP 800-106 [21]. We denote this variant of RMX by RMX_{SP}; its specification is placed in Appendix B. We observe the following differences between these two transforms:

1. RMX and RMX_{SP} differ in the padding rule defined to randomize the messages.
2. In RMX, the prepended random value of r is extended to a block of b bits by padding it with 0 bits (i.e. $r_0 = r \| 0^{b-|r|}$ such that $|r_0| = b$ bits). However, in RMX_{SP}, the prepended random value is directly concatenated with the exclusive-or of the message bits and the random value bits.

3.3. *Security Properties of TCR and eTCR Hash Functions*

The hash functions $\tilde{H}_r$ and H_r are eTCR and TCR respectively if their underlying compression function h is either chosen-SPR (c-SPR) or evaluated-SPR (e-SPR) [40]. Let t be the output size of h. These properties are defined below:

1. c-SPR: For a given message block m_i, find $(H_{i-1}, H_{i-1}^*, n_i)$ such that $h(H_{i-1}, m_i) = h(H_{i-1}^*, n_i)$.
2. e-SPR: Choose $u \geq 1$ values $\Delta_1, \ldots, \Delta_u$ each of length b bits. Receive a random value $r \in \{0,1\}^b$ and then define $m_i = r \oplus \Delta_u$ and $H_{i-1} = H^{H_0}(r \oplus \Delta_1 \| \ldots \| r \oplus \Delta_{u-1})$. Find (H_{i-1}^*, n_i) such that $h(H_{i-1}, m_i) = h(H_{i-1}^*, n_i)$.

Both these properties of h are related to its r-SPR property [40]. A generic birthday attack can be mounted on the c-SPR property of an ideal h, as an attacker can use the freedom in both the chaining values H_{i-1} and H_{i-1}^* as well as in the message block n_i to find a collision. Hence, the expected security level of the c-SPR property of h is about $2^{t/2}$. Due to the lack of freedom in varying (H_{i-1}, m_i), this attack does not apply to the r-SPR property of h (defined in Sect. 2.2). Therefore, the expected security level of r-SPR for an ideal h is about 2^t. Since an attacker has no control over the random value r used to compute the pair (H_{i-1}, m_i) in the e-SPR property of h, he also does not have the freedom to vary the pair (H_{i-1}, m_i) to apply a birthday attack on the e-SPR property of h. Therefore, the e-SPR property of h is much closer to the r-SPR property of h in its meaning, as noted in [40]. Therefore, one would expect that for an ideal h the security level of the e-SPR property is similar to the r-SPR property of h.

4. Generic Forgery Attacks on Randomize-Hash-then-Sign Signature Schemes

In Sect. 4.1, we present a generic existential forgery attack on the RMX-hash-then-sign signatures by Dang and Perlner [22] and discuss its limitations. In Sect. 4.2, we present a generic existential forgery attack on signature schemes based on H_r.

4.1. *Forgery Attack on the RMX-Hash-then-Sign Signature Schemes*

The online birthday forgery attack of [22] on signature schemes based on a t-bit RMX hash function requires $2^{t/2}$ chosen messages, $2^{t/2}$ offline hash function operations and a similar amount of memory. This attack is outlined below:

– *Online phase:*

1. Query the signer for the signatures of $2^{t/2}$ adaptive chosen messages m^i where $i = 1, \ldots, 2^{t/2}$. Store every m^i in a Table L_1.
2. The signer chooses a random value r_i to compute the signature s_i of every message m^i using SIG. The signer first computes $\mathrm{RMX}(m^i)$ and then computes $\mathsf{SIG}(H^{H_0}(\mathrm{RMX}(m^i))) = s_i$. The signer returns the pair (r_i, s_i) where $i = 1, \ldots, 2^{t/2}$. Store these pairs of values in L_1 corresponding to the chosen messages m^i.

– *Offline phase:*

1. Using r_i compute the hash values $H^{H_0}(\mathrm{RMX}(r_i, m^i))$ for $i = 1, \ldots, 2^{t/2}$ and store them together with (r_i, s_i) in L_1.
2. Choose random pairs (r_j, m_j) where j is in increments of 1 such that $(r_j, m_j) \neq (r_i, m^i)$ and compute the hash values $H^{H_0}(\mathrm{RMX}(r_j, m_j))$. While computing each of these hash values, check whether any of these hash values collide with any of the hash values in the Table L_1. After about $2^{t/2}$ trials, with a good probability, we can find a collision. Let that random pair be (r_y, m_y). That is, we can find (r_x, m^x, H_x, s_x) from L_1 where $(r_x, m^x) \neq (r_y, m_y)$ such that $H_x = H^{H_0}(\mathrm{RMX}(r_y, m_y)) = H^{H_0}(\mathrm{RMX}(r_x, m^x))$ and $\mathsf{VER}(r_y, m_y, H_y, s_x)$ outputs 1 where $x, y \in \{1, \ldots, 2^{t/2}\}$. Hence, $\mathsf{SIG}(m^x) = \mathsf{SIG}(m_y)$.
3. Output (m_y, r_y, s_x) as the forgery.

4.1.1. *Limitations of the Forgery Attack*

For extremely limited bandwidth applications, it is possible to save on the communication bandwidth by re-using the random value meant for the signing purposes as in RSA-PSS and DSS algorithms also as a salt for the RMX hash function mode. In the case of RSA-PSS, randomness used internally by the signature can be recovered by the recipient of the signature by using the RSA verification algorithm. In the case of DSS signatures, the random value used for the signature generation purpose is transmitted to the verifier along with the signature. These applications were also noted in [22,40].

We note that the generic forgery attack on the RMX-hash-then-sign signature scheme discussed in Sect. 4.1 does not succeed in these signature applications during the forgery verification step, as the random value used by the signer for RMX hashing and signing matches that of the arbitrary value chosen by the attacker with a negligible probability.

4.2. *Forgery Attack on Signature Schemes Based on H_r*

The offline phase of the generic forgery attack on the RMX-hash-then-sign signature schemes from Sect. 4 can be modified to forge signature schemes based on H_r. In the offline phase, the attacker computes hash values for some chosen messages under H^{H_0}

until he finds a candidate message, say N, whose hash value $H^{H_0}(N)$ collides with the randomized hash value of one of the queried messages, say m^x, computed during the online phase of the attack. Let this randomized hash value be $H_{r_x}(m^x)$ and the signature of $H_{r_x}(m^x)$ be s_x. The attacker then computes $m^* = N \oplus r_x$ and produces (m^*, r_x, s_x) as the forgery. This attack does not produce a meaningful message as the forgery because m^* is random and does not render any meaning.

Remark 3. The number of queries to the signer required for the generic forgery attacks in Sects. 4.1 and 4.2 can be reduced by increasing offline compression function computations such that the product of the number of offline computations and online queries is equal to 2^t. In addition, these attacks are independent of the size of the random value used in the randomized hashing and length of the messages queried to the signer.

Remark 4. The *online phase* of the forgery attacks outlined in Sects. 4.1 and 4.2 is also possible in the *known message attack model*, leading to more powerful forgery attacks. That is, an attacker can use only known messages and their signatures instead of *adaptively* querying chosen messages to the signer for signatures.

5. Application of Fixed Point Expandable Message to Forge Hash-then-Sign Signature Schemes

5.1. *Fixed Point Expandable Message*

Dean [25] showed that if it is easy to find fixed points (see Sect. 2.4) for a compression function, then a multicollision based on several distinct length messages can be constructed for the hash function in about twice as much as the work required to build a birthday collision attack. Kelsey and Schneier [46] called a multicollision where messages of different lengths collide to the same hash value an *expandable message*. An *expandable message* does not yield a collision to the full hash function due to the provision of the encoding of the length of the messages in the last block. This *expandable message* attack on the hash function H^{H_0} based on the Davies–Meyer compression function h and without message length encoding is outlined below:

1. For $i = 1, \ldots, 2^{t/2}$, construct $2^{t/2}$ random fixed points (H_{i-1}, n_i) for h. That is, $h(H_{i-1}, n_i) = H_{i-1}$. Store these fixed points in a Table L_1.
2. Build $2^{t/2}$ hash values from the initial state H_0 of H^{H_0} by processing that many message blocks and store these blocks and hash values in a Table L_2.
3. Find a match between the hash values stored in Tables L_1 and L_2 and let n and m be the corresponding message blocks. This implies a collision of form $H^{H_0}(m) = H^{H_0}(m\|n)$.
4. Output $(m, m\|n)$ as the *expandable message* where two messages of lengths 1 and 2 blocks produced a collision.[3]

The complexity of this attack is about $2^{t/2+1}$ compression function computations and a similar amount of memory.

[3] An *expandable message* of desired size can be obtained by appending a desired number of copies of the fixed point message block n to $m\|n$.

5.2. *Forgery Attack on Hash-then-Sign Signatures*

We show a forgery attack on hash-then-sign signature schemes by employing a variant of Dean's technique of finding fixed point *expandable messages*. In this attack we ensure that the encoding of the length of the message to be produced as a forgery is integrated in the fixed point block and build a collision of form $H^{H_0}(m) = H^{H_0}(m\|n)$. We then query the signer for a signature on m and show that $\mathsf{SIG}(m\|n) = \mathsf{SIG}(m)$, thereby showing $m\|n$ as the forgery of m. The complexity of the attack is $2^{t/2+1}$ computations of H^{H_0} and one query to the signer. Let h be the Davies–Meyer compression function. This forgery attack is outlined below:

1. Consider $2^{t/2}$ messages m^i where $i = 1, \ldots, 2^{t/2}$. Let these messages contain c number of b-bit blocks and let the length of these messages in bits be $(c \times b) - (l+1)$. Pad each m^i with a bit 1 followed by l bits that represent the binary format of the length $(c \times b) - (l+1)$ bits. Compute the hash values H_i for each of these padded messages under H^{H_0}. Store m^i and H_i in a Table L_1.
2. For $j = 1, \ldots, 2^{t/2}$, compute $2^{t/2}$ fixed points for h such that $h(H_j^*, n^j) = H_j^*$ where the last $l+1$ bits of every block n^j are fixed with a padding bit 1 and l bits that represent the binary format of the length of the message $m^i\|pad\|(n^j)^{[b-(l+1)]}$ where $(n^j)^{[b-(l+1)]}$ represents the first $b-(l+1)$ bits of n^j. Let the last $l+1$ bits of n^j be a string called *padf* which means padding bits in the fixed point block. Store H_j^* and $(n^j)^{[b-(l+1)]}$ in a Table L_2.
3. Due to the birthday paradox, with a significant probability we can find a hash value H_x from the Table L_1 and a hash value H_y^* from the Table L_2 such that $H^{H_0}(m^x\|pad) = H_x = H_y^* = h(H_y^*, n^y)$ for some $x \in \{1, \ldots, 2^{t/2}\}$ and $y \in \{1, \ldots, 2^{t/2}\}$.

   This implies $H^{H_0}(m^x\|pad\|n^y) = H^{H_0}(m^x\|pad) = H_x$. Let $m = m^x$ and $n = (n^y)^{[b-(l+1)]}$. Therefore, $H^{H_0}(m\|pad\|n\|padf) = H^{H_0}(m\|pad)$. Recall that *padf* bits are the padding bits of the message $m\|pad\|n$ under H^{H_0}.
4. Query the signer for the signature on the message m. The signer hashes the message $m\|pad$ using H^{H_0} and then signs the hash value $H^{H_0}(m\|pad)$ using the signature algorithm SIG to obtain the signature $s = \mathsf{SIG}(m)$.
5. Since $H^{H_0}(m\|pad\|n\|padf) = H^{H_0}(m\|pad)$, $\mathsf{SIG}(m\|pad\|n) = \mathsf{SIG}(m)$.
6. Output $(m\|pad\|n, s)$ as the forgery.

Remark 5. Our attack technique subtly differs from Dean's method [25], as we exert control over the fixed point message blocks by integrating the padding and length encoding bits that represent the length of the forgery message in the last few bits of the fixed point block. This is in contrast to Dean's method of finding *expandable messages* where all bits in the fixed point block are random. In addition, our forgery attack does not use an *expandable message*, as our attack requires a collision for two distinct length messages for the full RMX hash function. Therefore, our attack technique is stronger than the one used by Dean; hence our technique would also be applicable to build *expandable messages* for the hash functions when the message inputs are exclusive-ored with a random value as in RMX.

6. Offline Collision Attacks Are not Useful to Forge Randomize-Hash-then-Sign Signatures

We argue that offline collision attacks that are used to forge hash-then-sign digital signatures such as the one outlined in Sect. 5 are not useful 'as is' to forge the RMX-hash-then-sign digital signatures. The reason is that a signer who uses the RMX-hash-then-sign signature algorithm to sign the message m chooses a salt r and signs the hash value $\tilde{H}_r^{H_0}(m)$ to obtain the signature s. The signer then sends the triplet (m, r, s) to the receiver. Therefore, to forge an RMX-hash-then-sign algorithm, we must satisfy the condition that the offline birthday collision obtained for the hash function H^{H_0} in Step 3 of the forgery attack in Sect. 5 would also produce a collision for the RMX hash function mode $\tilde{H}_r^{H_0}$. That is, we must obtain a condition $\tilde{H}_r^{H_0}(m\|pad\|n\|padf) = H_r^{H_0}(m\|pad)$ to claim $m\|pad\|n$ as the forgery message of m under the signature algorithm SIG. However, this condition holds with a probability of 2^{-t}, as the collision obtained in the offline phase of the attack would no longer exist once the two colliding messages $m\|pad\|n$ and m are randomized with the salt r. Hence, we have to find a collision for the messages that have already been randomized with the salt r to be able to forge RMX-hash-then-sign signatures. Since the signer chooses a fresh salt at random for every signature generation, we must be able to predict this salt to find a collision for the messages randomized with the salt. This is an improbable event for a salt r with a typical size such as $|r| \in [128, b]$ where $b = 512$ bits for compression functions such as those in the MD5 and SHA-1 hash functions.

These security arguments also apply when one attempts to forge TCR-hash-then-sign signatures by using offline collision attacks. Therefore, offline collision attacks are not useful to forge randomize-hash-then-sign algorithms. These attacks must be inherently online, as shown in the generic forgery attacks on the randomize-hash-then-sign algorithms in Sect. 4 and on some RMX-hash-then-sign algorithms in Sect. 7.

7. Existential Forgery Attack on Some RMX-Hash-then-Sign Signatures

We extend the technique presented in Sect. 5 to provide a new existential forgery attack on the RMX-hash-then-sign signatures that use compression functions for which fixed points can be easily found. The significance of this new attack is that it can be used to forge RMX-hash-then-sign signature schemes (e.g. RSA-PSS and DSS schemes based on RMX hash) when the same salt is used for both RMX hashing and signing; thus overcoming the limitation of the generic forgery attack on the RMX-hash-then-sign signatures presented in Sect. 4.1. We first provide an outline of the attack and then provide its pseudocode.

7.1. *Outline of the Forgery Attack*

In this attack, we first determine the lengths of the messages to be forged and to be produced as a forgery. We then compute $2^{t/2}$ fixed point message blocks for the compression function by integrating padding bits (including length-encoded bits) required for the forgery message into the fixed point blocks. We then query the signer to sign $2^{t/2}$

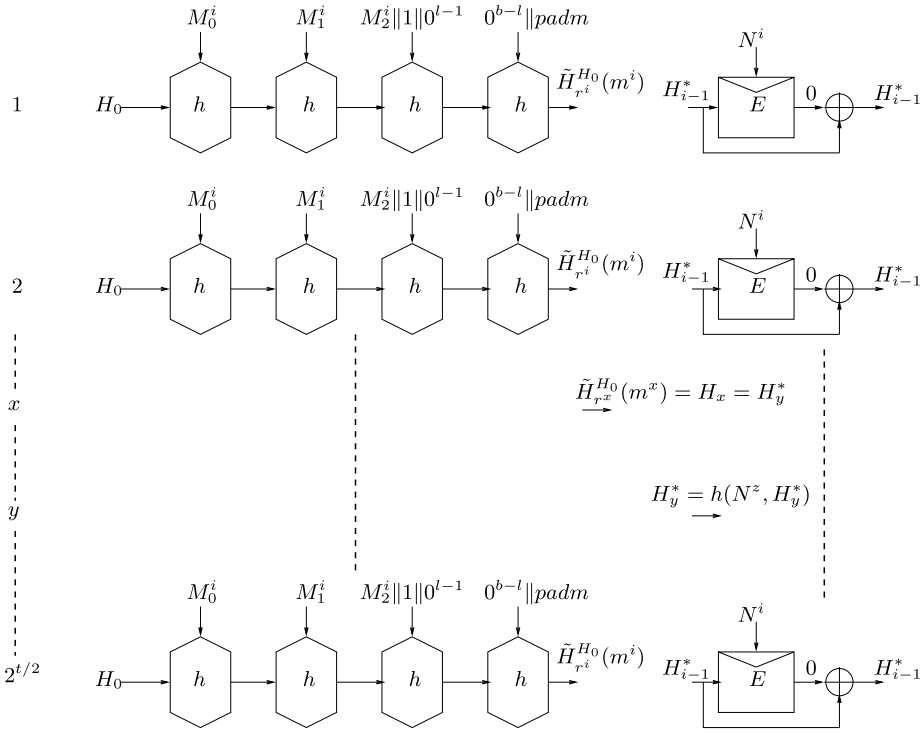

Fig. 3. Illustration of birthday collision attack between the hash values of RMX hash mode $\tilde{H}_r$ and fixed point chaining values of the compression function h. RMX hash values and fixed points are built for $i = 1, \ldots, 2^{t/2}$. The message block m^x under the RMX hash mode $\tilde{H}_r$ with the hash value H_x is shown as colliding with the chaining value H_y^* of a fixed point block N^z.

equal length adaptive chosen messages[4] and collect their signatures and random values used for the generation of the signatures. We use these $2^{t/2}$ random values and messages to compute $2^{t/2}$ RMX hash values and find an RMX hash value which collides with one of the $2^{t/2}$ pre-computed fixed point hash values. This attack is illustrated in Fig. 3. Due to the birthday paradox, with a good probability, we can find a chosen message (along with its random value and signature) whose RMX hash value collides with the hash value of one of the fixed point message blocks. This technique is illustrated in Fig. 3 and in Fig. 4(a) and (b). We then use the exclusive-or operation to mix the fixed point block and the random value used to sign the chosen message to obtain a new block as shown in Fig. 4(c). Finally, we concatenate this block to the chosen message and produce this concatenated message as the forgery of the chosen message. The attack involves some subtleties due to the fact that the RMX hash function has a padding layer as specified in Sect. 3.2.1 and, hence, the attack has an additional negligible complex-

[4] Although we must make queries of the same length, they may still be chosen adaptively respecting this length constraint.

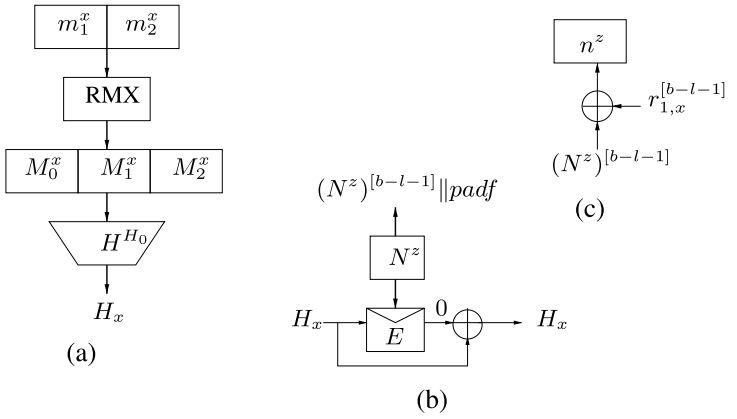

Fig. 4. Illustration of the forgery attack on the RMX-hash-then-sign scheme based on Davies–Meyer.

ity of 2^{24} which adds to the complexity of $2^{t/2}$ adaptive chosen messages and $2^{t/2+1}$ operations of the compression function.

7.2. *Pseudocode of the Forgery Attack*

The pseudocode for the existential forgery attack on the signature scheme SIG which uses the RMX hash function $H_r^{H_0}$ based on the Davies–Meyer compression function h is given below:

– *Pre-computation phase:*

1. Determine the length of the message to be forged. Let m be a message whose signature has to be forged and let the size of m be $2b-l-24$ bits. Let m^* be the forgery of m to be produced whose length can be pre-determined using $|m|$. That is, $|m^*| = |m| + l + 24 + b + (b-l-24-1) = 4b-l-25$ bits.
2. Compute $2^{t/2}$ fixed points for the compression function h by using message blocks N^j, each of size b bits, such that $H_{j-1}^* = h(H_{j-1}^*, N^j)$ for $j = 1, \ldots, 2^{t/2}$. While finding fixed points, fix the last $l+1$ bits of each message block N^j with the pad bit 1 and l bits to represent the pre-determined length encoding of the message m^* of length $4b-l-25$ bits to be produced as the forgery. Let these $l+1$ bits be *padf* bits. Store the pairs (N^j, H_{j-1}^*) in a Table L_1 where $j = 1, \ldots, 2^{t/2}$.

– *Online phase:*

1. For $i = 1, \ldots, 2^{t/2}$, query the signer with equal length adaptive chosen messages[5] m^i. Let $|m^i| = b + b - l - 24$ bits. Every message m^i can be represented as $m^i = m_1^i \| m_2^i$ where $|m_1^i| = b$ bits and $|m_2^i| = b-l-24$ bits. Store these $2^{t/2}$ messages in a Table L_2.

[5] All queried messages can also be the same. Since the signer always chooses a fresh salt to randomize each queried message independently of the value of the message, all randomized messages are distinct.

For $i = 1, \ldots, 2^{t/2}$, the signer computes the signatures s_i on the equal-length messages m^i as follows:

(a) The signer chooses a fresh random value r_i for every ith query independent of the message m^i. The signer calculates three random values $r_{0,i}$, $r_{1,i}$ and $r_{2,i}$ for every chosen random value r_i following the RMX specification in Sect. 3.2.1 and Remark 1 as follows:
   i. $r_{0,i} = r_i \| 0^{b-|r_i|}$
   ii. $r_{1,i} = r_i \| r_i \| \ldots \| r_i$ such that $|r_{1,i}| = b$ bits and the last repetition of r_i is truncated if needed.
   iii. $r_{2,i} = r_{1,i}^{[b-l]}$ (as noted in Remark 1).

(b) The signer divides every message m^i as $m^i = m_1^i \| m_2^i$ where $|m_1^i| = b$ bits and $|m_2^i| = b - l - 24$ bits.

(c) The signer randomizes every message m^i as follows:
   i. $M_0^i = r_{0,i}$
   ii. $M_1^i = m_1^i \oplus r_{1,i}$
   iii. $M_2^i = (m_2^i \| 0^8 \| lpad) \oplus r_{2,i}$

(d) Let *padm* represent the l-bit binary encoded format of the length of the message $M_0^i \| M_1^i \| M_2^i$. For every randomized message $M_0^i \| M_1^i \| M_2^i$, the signer computes the hash value by passing this message as an input to the hash function H^{H_0}. The hash value for every randomized message is $\tilde{H}_{r_i}^{H_0}(m^i) = H^{H_0}(M_0^i \| M_1^i \| (M_2^i \| 1 \| 0^{l-1}) \| (0^{b-l} \| padm))$. The signer returns the signature $s_i = \mathsf{SIG}(\tilde{H}_{r^i}^{H_0}(m^i))$ of every message m^i.

2. For every queried message m^i where $i = 1, \ldots, 2^{t/2}$, the signer returns the pair (r_i, s_i).

– *Offline phase:*

1. Add the received pair (r_i, s_i) to the Table L_2.
2. Using the random value r_i, compute three random values $r_{0,i}$, $r_{1,i}$ and $r_{2,i}$ following the RMX specification in Sect. 3.2.1 and Remark 1.
3. Randomize the message m^i as follows:
   (a) $M_0^{i'} = r_{0,i}$
   (b) $M_1^{i'} = m_1^i \oplus r_{1,i}$
   (c) $M_2^{i'} = (m_2^i \| 0^8 \| lpad) \oplus r_{2,i}$
4. Let $M^{i'} = M_0^{i'} \| M_1^{i'} \| M_2^{i'}$. The validity of the signatures s_i returned by the signer during the *online phase* of the attack ensures that $M_0^{i'} \| M_1^{i'} \| M_2^{i'} = M_0^i \| M_1^i \| M_2^i$.
5. Compute $\tilde{H}_{r_i}^{H_0}(m^i) = H^{H_0}(M_0^i \| M_1^i \| (M_2^i \| 1 \| 0^{l-1}) \| (0^{b-l} \| padm))$ and add the hash values $\tilde{H}_{r_i}^{H_0}(m^i)$ to the Table L_2. Let $\tilde{H}_{r_i}^{H_0}(m^i) = H_i$ for $i = 1, \ldots, 2^{t/2}$. Now the Table L_2 contains the values (m^i, r_i, s_i, H_i).
6. Find a collision between the $2^{t/2}$ hash values stored in the Tables L_1 and L_2. This step is illustrated in Fig. 3. With a good probability, we can find a hash value H_x from the Table L_2 and a hash value H_y^* from the Table L_1

such that:

$$H^{H_0}\left(M_0^x \| M_1^x \| \left(M_2^x \| 1 \| 0^{l-1}\right) \| \left(0^{b-l} \| padm\right)\right) = H_x = h\left(H_y^*, N^{y+1}\right) = H_y^*$$

where
(a) $M_0^x = r_{0,x}$ and $|M_0^x| = b$
(b) $M_1^x = m_1^x \oplus r_{1,x}$ and $|M_1^x| = b$
(c) $M_2^x = (m_2^x \| 0^8 \| lpad) \oplus r_{2,x}$ and $|M_2^x| = b - l$
(d) $N^{y+1} = (N^{y+1})^{[b-l-1]} \| padf$
and $x \in \{1, \ldots, 2^{t/2}\}$, $y \in \{0, \ldots, 2^{t/2} - 1\}$. This step is illustrated in Fig. 3 and in Fig. 4(a) and 4(b). Let $y + 1 = z$.

7. Let $r'_{2,x}$ be the last l bits of $r_{1,x}$. Then $r_{1,x} = r_{2,x} \| r'_{2,x}$.
   Note that $|r_{2,x}| = b - l$. Let $padr = r'_{2,x} \oplus (1 \| 0^{l-1})$.
   Let $padr1 = (r_{1,x}^{[b-l]} \oplus 0^{b-l}) \| (r'_{2,x} \oplus padm)$. Note that $|padr1| = b$ bits.
8. Calculate $(N^z)^{[b-l-1]} \oplus r_{1,x}^{[b-l-1]} = n^z$ as shown in Fig. 4(c). The probability that the last 24 bits of n^z are $0^8 \| lpad$ is 2^{-24}. These 24 bits represent the padding bits of the message randomized by RMX.
9. Let $m^* = m_1^x \| (m_2^x \| 0^8 \| lpad \| padr) \| padr1 \| (n^z)^{[|n^z|-24]}$ and $m = m_1^x \| m_2^x$. Note that $|m| = 2b - l - 24$ bits and $m^* = 4b - l - 25$ bits, which is the same as determined in Step (1) of the *Pre-computation phase* of the attack. The signature $\mathsf{SIG}(m)$ on the message m is also valid on m^* as $\tilde{H}_r^{H_0}(m) = \tilde{H}_r^{H_0}(m^*)$. Let $\mathsf{SIG}(m) = s$.
10. Finally, output (m^*, r, s) as the forgery.

7.2.1. *Complexity*

It takes $2^{t/2}$ operations of h to compute the fixed points, $2^{t/2}$ adaptive chosen message queries to the signer during the *online phase*, $2^{t/2}$ operations of h and $2^{t/2}$ exclusive-or operations during the *offline phase* and a probability of 2^{-24} to match the correct padding bits of the RMX transform. Therefore, the computational complexity of the attack is approximately $2^{t/2+1}$ operations of the compression function and $2^{t/2}$ chosen messages. The memory requirements of the attack are as follows assuming that the size of the signature is four times the security level of $t/2$ bits of a t-bit hash (which is possible for the DSA algorithm): $b \times 2^{t/2} + t \times 2^{t/2}$ bits in the *pre-computation phase*, $2b \times 2^{t/2}$ bits in the *online phase* and $|r| \times 2^{t/2} + t \times 2^{t/2} + 2t \times 2^{t/2}$ bits in the *offline phase*. The memory required for the attack is equal to $(3b + 4t + |r|) \times 2^{t/2}$ bits. This is approximately $2^{t/2+3}$ memory of t-bit values.

7.3. *Practical Security*

Here we illustrate with an example the practical security offered by the RMX-hash-then-sign signature algorithms. Our attack can be used to forge a signature scheme based on RMX-SHA-256 in about 2^{129} operations of the SHA-256 compression function, 2^{128} chosen messages and a probability of 2^{-24} for the forged message to contain the 24 padding bits used in the RMX transform and a memory of about 2^{131}. In addition, our attack is independent of the size of the random value r. Note that in our attack we

assumed that $b' = b - l - 24$ bits. The attack also applies for about the same complexity when $b' < b - l - 24$ bits. However, when $b' > b - l - 24$ bits, the padding bits of the RMX transform are placed in the last two blocks M_L and M_{L+1}. Since the last block M_{L+1} does not contain any message bits, it is not possible to generate a fixed point block which can represent this block; hence the attack does not apply.

In comparison, as remarked in [40], forging a signature scheme based on RMX-SHA-256 by using the long message second preimage attack of [25,46] requires about 2^{201} offline operations of the SHA-256 compression function, more than 2^{55} memory and one chosen message of size about 2^{55} message blocks each of 512 bits. We briefly outline this attack by assuming $|r| = b$ bits for an easy description.

1. Query the signer with a long message $m = m_1 \| \ldots, m_{L-1}$ where $L = 2^{55}$, $|m_i| = 512$ bits and the block m_L is reserved for the padding bits required by the SHA-256 algorithm. The signer produces the signature s for the hash value $H^{H_0}(\text{RMX}(r, m))$ by using the signature algorithm SIG and sends (r, s) to us.
2. Compute $H^{H_0}(r)$.
3. Use $H^{H_0}(r)$ as the starting state of the hash function and find a second preimage for m by following the Kelsey–Schneier [46] long message second preimage attack algorithm. Let $n = n_1 \| n_2, \ldots, \| n_{L-1}$ be the second preimage which has the same size as m.
4. Compute $m^* = n_1 \oplus r \| n_2 \oplus r, \ldots, \| n_{L-1} \oplus r$.
5. Now, $H^{H_0}(\text{RMX}(m, r)) = H^{H_0}(\text{RMX}(m^*, r))$ and produce m^* as the second preimage of m under the hash function $H^{H_0}(\text{RMX}(.))$. Therefore, the signature s computed for the message m by using SIG based on RMX-SHA-256 is also valid for m^*.
6. Output (m^*, r, s) as the forgery.

The time complexity of this attack increases when we desire to produce forgeries for messages of size less than 2^{55} blocks. For example, the complexity of this attack to produce a forgery message of size 512 bits will be the same as the brute force attack complexity of 2^{256} SHA-256 operations.

Remark 6. We can also derive trade-offs between the online and offline time complexities for the forgery attack in Sect. 7 similar to the generic forgery attack on the RMX-hash-then-sign algorithms discussed in Sect. 4. In addition, a variant of our attack is possible wherein an attacker can query several signers but with less than $2^{t/2}$ queries to each of them and finally forge the signature of one of the signers, one of whose randomized hash values collides with one of the pre-computed hash values.

Remark 7. Unlike the generic forgery attacks of Sect. 4, the forgery attack of Sect. 7 is not applicable in the *known message attack* model unless the lengths of all $2^{t/2}$ known messages are the same.

7.4. *Attack on the e-SPR Property of the Compression Functions*

Our forgery attack on the RMX-hash-then-sign signature schemes translates into a birthday collision attack on the e-SPR property of the compression function h for which

fixed points can be easily found. Recall that in the e-SPR game, we choose $u \geq 1$ values $\Delta_1, \ldots, \Delta_u$, each of length b bits. We then receive a random value $r \in \{0,1\}^b$ and define $m_i = r \oplus \Delta_u$ and $H_{i-1} = H^{H_0}(r \oplus \Delta_1 \| \ldots \| r \oplus \Delta_{u-1})$. Finally, we aim to find a pair (H^*_{i-1}, n_i) such that $(H^*_{i-1}, n_i) \neq (H_{i-1}, m_i)$ and $h(H_{i-1}, m_i) = h(H^*_{i-1}, n_i)$. The attack is outlined below:

1. Find $2^{t/2}$ fixed points (H_{i-1}, m^i) for h where $i = 1, \ldots, 2^{t/2}$ and store them in a Table L.
2. Play the e-SPR game for $2^{t/2}$ times with $u = 2$ where $\Delta_1 = \Delta_2 = 0$. For every call to the e-SPR game, we receive a fresh random value r_j for $j = 1, \ldots, 2^{t/2}$.
3. Due to the birthday paradox, we can find $H_{i-1} = H^{H_0}(r_j \| r_j)$ for some i and j with a good probability. Let this $r_j = r$ and m^i in the fixed point (H_{i-1}, m^i) be m_i for some i.
4. Let $H^*_{i-1} = H^{H_0}(r)$, $n_i = r$, $H_{i-1} = H^{H_0}(r \| r)$.
   Now, we have $h(H^*_{i-1}, n_i) = H^{H_0}(r \| r) = H^{H_0}(r \| r \| m_i) = h(H_{i-1}, m_i)$.

Thus, after $2^{t/2}$ games, we expect to win one game. Note that the forgery attack in Sect. 4.1 also translates into an e-SPR attack on any compression function after $2^{t/2}$ e-SPR games.

These attacks on the e-SPR property of the compression function illustrate that the security level of e-SPR is not as high as expected (i.e. similar to the second preimage resistance) when an attacker plays the e-SPR game for $2^{t/2}$ times. In contrast, the security reduction of [40] from the e-TCR property of the RMX hash function mode to the e-SPR property of the compression function considers an attacker who plays the e-SPR game only once (i.e. the attacker receives random value r only once). Hence, our analysis complements the analysis of [40] by considering an attacker who can play the e-SPR game more than once.

8. Applications of Our Forgery Attack

In this section, we discuss some interesting applications of our forgery attack on the RMX-hash-then-sign digital signatures.

8.1. *Meaningful Forgeries*

The method used to forge RMX-hash-then-sign schemes provided in Sect. 7 may be used to produce an 'almost' meaningful forgery message by querying meaningful messages to the signer during the *online phase* of the attack. Then all bits in the forgery message except those in the fixed point block convey the meaning, and all bits in the fixed point block except the padding bits are random. In addition, this method is also applicable to forge signature schemes based on the TCR hash function mode on which the generic attack of Sect. 4.2 does not produce a meaningful forgery.

8.2. RMX_{SP} *is Weaker than RMX*

NIST standardized a variant of the RMX hash function in SP 800-106 [21]. We called this variant RMX_{SP} in Sect. 3.2.2 and it is defined in Appendix B. Our forgery attack

on the RMX-hash-then-sign algorithms also extends to those based on RMX_{SP} that use fixed point compression functions. When $|m| + 1 \geq |r|$ for RMX_{SP}, the complexity of our forgery attack on the SIG based on RMX_{SP} is similar to the one on the SIG based on RMX with the exception that it requires a success probability of $1/2$ to hit the correct padding bit '1' used to pad the message by RMX_{SP}. Note that RMX uses 24 bits for padding; hence our attack requires 2^{-24} success probability to match the padding bits of RMX correctly. Hence, RMX_{SP} is weaker than RMX due to its weak padding compared to the one in RMX.

8.3. *Variants of Merkle–Damgård*

Our forgery attack is also applicable on signature schemes based on a variant of the eTCR hash function mode $H^{H_0}(r \| H_r^{H_0}(m))$ proposed by Halevi and Krawczyk [40]. It is also applicable to signature schemes that use the RMX hash function together with the split padding [87] technique used to pad the message input to the hash function. The split padding employed for a hash function ensures that a minimum number of message bits are used in every block including the padding and message length encoding blocks. The forgery attacks on the RMX-hash-then-sign algorithms outlined in Sect. 7 and in Sect. 4 are also applicable to signature schemes based on the hash functions with sequential counters (for example, HAIFA hash function mode [13]) as the counter inputs can be controlled in both the attacks. However, sequential counters for the RMX hash function would still prevent the attempts to forge RMX-hash-then-sign schemes by using the long message second preimage attack of Kelsey-Schneier [46] as recalled in Sect. 7.3.

Remark 8. Our online birthday forgery attacks are not useful to forge signature schemes that use the randomized setting of the wide-pipe hash construction [51] with an internal state size $w \geq 2t$ as the attack requires at least 2^t chosen messages and 2^{t+1} operations of the compression function. This remark also holds even if it is easy to find fixed points for the compression functions used in the wide-pipe hash function. For example, the online birthday forgery attacks do not apply to the signatures based on the RMX hash mode of Grøstl [36], a finalist in the NIST's SHA-3 hash function competition, which uses a compression function for which many fixed points can be easily found but has $w \geq 2t$ for a t-bit hash value.

9. On the Applicability of Our Analysis to the Hash Functions with 'Built-in' Randomization

Consider a Merkle–Damgård hash construction in which salt is mixed with the input state and message block inside the compression function at every iteration. Such constructions are called hash functions with 'built-in' randomization. This method of randomizing hash functions is different from the setting we have analyzed in this paper. The generic forgery attack on the RMX-hash-then-sign schemes presented in Sect. 4.1 is also applicable to signature schemes based on this randomized hash function setting. However, the applicability of the forgery attack on signature schemes using TCR collisions as presented in Sect. 4.2 to those based on these functions depends on how the random value is mixed with the message and state. For the same reason, even if the

compression functions have fixed points, the techniques presented in Sect. 7 are not directly applicable to forge signature schemes based on hash functions with 'built-in' randomization. Hash function designs such as LAKE [6], BLAKE [5] and ECHO [11] have this kind of 'built-in' support for randomization. Moreover, constructing an eTCR attack using certain types of collision attacks on the hash functions with 'built-in' randomization that rely on the fixed difference in the salt inputs as well as on their actual values such as on LAKE [15] depends on the distribution of the salt inputs used in the collision attack. The reason is that the choice of one of the salts in the eTCR attack on a randomized hash function is determined by the signer and is not under the control of the attacker. Analyzing eTCR and TCR properties of hash functions with built-in support for randomization requires analysis of the c-SPR and e-SPR properties of the compression functions.

10. Randomize-Hash-then-Sign Signatures Versus HMAC

Recall that the random value chosen by the signer is unknown to the attacker who tries to break the TCR/eTCR properties of the randomized hash functions, and the signer chooses a fresh random value for each query. This property is similar to the security requirement of message authentication codes (MACs) in which the task of an attacker is to forge a MAC function without knowledge of the secret key. Moreover, the standard MAC function HMAC [7,8] has been proposed as a safety net for the MAC functions that depend on the hashing strength as little as possible [42], just like randomized hash functions. These similarities in the design goals and security properties of randomized hash functions and HMAC makes it interesting to compare the attack models on them when they are instantiated with the same ideal compression function.

Offline birthday collision attacks on the hash functions do not help to forge HMAC [8]. The best known forgery attack on HMAC is the online birthday collision attack [8]. In this attack on a t-bit HMAC, the attacker first finds two distinct messages m and m^* such that $\text{HMAC}(m) = \text{HMAC}(m^*)$ by querying the HMAC oracle with at most $2^{t/2}$ chosen messages. Then he queries the HMAC oracle for the tag of the message $m\|n$ and produces the message $m^*\|n$ as the forgery of $m\|n$ since $\text{HMAC}(m\|n) = \text{HMAC}(m^*\|n)$ due to the length extension property of the Merkle–Damgård hash functions. It is not possible to derive online and offline computation time trade-offs for this attack, unlike for the forgery attacks presented on randomize-hash-then-sign schemes. While it is possible to generate multiple forgeries for HMAC after querying its oracle with $2^{t/2}$ chosen messages [16,53], an online birthday attack on the randomize-hash-then-sign signature does not lead to more than one forgery.

The cryptanalytic (second) preimage attacks on the hash functions such as those on MD4 [50], MD5 [77], AURORA-512 [32,75,76] and the reduced round version of SHAvite-3-512 [37] directly extend to finding second preimages for their randomized hash function modes; hence these attacks lead to forgery attacks on signature schemes based on these hash functions with just one chosen message. However, the possibility of the forgery or key recovery attacks on HMAC using cryptanalytic (second) preimage attacks on the hash functions depends on the subtle details of the attacks and does not extend to the attacks on HMAC with just one chosen message query. Moreover, some cryptanalytic collision attacks on the hash functions can be extended to forge

HMAC and its derivative NMAC functions; of course, this depends on the subtle details of the collision attacks. See [19,33,47,69,70,86] for the analytical techniques used in the forgery and key recovery attacks on HMAC and NMAC functions based on some popular hash functions such as MD5 and SHA-1.

11. Conclusion and Open Questions

In this article, we presented several existential forgery attacks in an online setting on the digital signatures based on the two randomized hash functions proposed by Halevi and Krawczyk [40] and the signatures based on the RMX variant standard of NIST in SP 800-106. All our attacks require a large number of queries to the message signer as in a traditional birthday attack on the size of the hash value. Our analysis shows that the randomized hash function standardized by NIST is slightly weaker than the RMX hash mode proposed by Halevi and Krawczyk. However, there has been no change to the standard since the discovery of our attacks.

Our results are in no contradiction with the previous analysis of Halevi and Krawczyk [40], who proved that the problem of forging these signatures reduces to a problem similar to that of finding second preimages for the underlying hash function in these algorithms. However, our results demonstrate that the security of randomize-hash-then-sign signature algorithms does not necessarily rely on the second preimage resistance of the hash functions, as better attacks can be found in an online model. It is clear from our attacks and those of [25,46] that it is well worth investigating the second preimage resistance properties of the compression functions to identify possible weaknesses that may affect the randomized hashing setting.

Our work opens an interesting research problem on how to improve the design of randomized hash function modes so that the signatures based on these modes provide about 2^t security in the online setting and require no changes to the current standard hash functions (e.g. SHA family) and signature schemes. It is also interesting to design novel block cipher-based compression function structures that can avoid attacks on hash functions and their applications through undesirable properties such as fixed points. For instance, it is an interesting research problem to design and formally analyze block cipher-based compression functions that are indifferentiable in the ideal cipher model from a random oracle whose inputs are of fixed length.

Acknowledgements

1. Praveen Gauravaram is sponsored by the Danish Council for Independent Research – Technology and Production Sciences (FTP) grant number 09-066486/FTP.
2. The research work presented in this paper has been supported in part by the European Commission through the ICT programme under contract ICT-2007-216676 ECRYPT II.
3. We thank the anonymous reviewers of the Journal of Cryptology for their several valuable comments and suggestions to improve the presentation, technical reasoning and editorial quality of the submitted version of the article. We also

thank anonymous reviewers of EUROCRYPT 2009 for comments on the version submitted to the EUROCRYPT 2009 conference. We also thank Quynh Dang, Pierre-Alain Fouque, Choudary Gorantla, Shai Halevi, Guo Jian, Hugo Krawczyk, Gregor Leander, Chris Mitchell, Ray Perlner, Yu Sasaki and Erik Zenner for discussions on the subject of randomized hash functions.

Appendix A. Observation in the Padding Rule of RMX

Recall from Sect. 3.2.1 that the RMX transform appends k zero bits and a 16-bit string called *lpad* to the message bits in a block m_L of size b' which is less than the block size b of the compression function. We noted that when $b' \leq b - l - 24$, the value of k determines whether an additional b-bit block m_{L+1} must be appended to m_L to accommodate the padding and l-bit binary encoding of the true length of the message. Here we illustrate this property for the RMX-SHA-256 construction.

Consider hashing of a message m using RMX-SHA-256. For SHA-256, $b = 512$ bits. Let $|m| = 512 + 424 = 936$ bits, where $|m_1| = 512$ and $|m_2| = 424$ bits. Following RMX specification from Sect. 3.2.1, $b' = b - l - 24 = 512 - 64 - 24 = 424$ bits and $k = b - b' - 16 - l = 512 - 424 - 16 - 64 = 8$ bits. Let $|r| = 128$ bits. Now the randomized message M is defined as follows:

1. $M_0 = r_0$
2. $M_1 = m_1 \oplus r_1$
3. Calculation of M_2:
   (a) $M_2^* = m_2 \| 0^8 \| lpad$ where $|M_2^*| = 448$ bits
   (b) $M_2 = M_2^* \oplus r_2$

Therefore, $\text{RMX}(r, m) = M = M_0 \| M_1 \| M_2$. A hash function H^{H_0} used to process M requires at least $l + 1$ bits for padding and encoding the length of the message. For SHA-256, $l = 64$ bits, and hence it requires at least 65 bits for padding and message length encoding. It is difficult to accommodate more than 64 bits in the remaining l-bit positions in the last block M_2, as it already has 448 bits. Therefore, to process M using SHA-256, M is padded as follows: $M = M_0 \| M_1 \| (\underbrace{M_2 \| 1 \| 0^{63}}_{512 \text{ bits}}) \| \underbrace{(0^{448} \| l)}_{512 \text{ bits}}$ where l represents the 64-bit binary encoded format of the length of M. Similarly, if $b' = 423$ bits, then $k = 9$ bits and $M_2^* = m_2 \| 0^9 \| lpad$. So, if $b' \leq b - l - 24$ then H^{H_0} requires an extra block to pad and to encode the length of M.

Alternatively, when $b' \leq b - l - 24$ bits, we could define $k = b - b' - 24 - l$ bits. Then the hash function H^{H_0} does not require an extra block to encode the length of the message M. In the above illustration, when $|m| = 936$ bits, $M_2^* = m_2 \| 0^0 \| lpad$ and $M_2 = M_2^* \oplus r_2$ where $|M_2^*| = 440$ bits and $M = M_0 \| M_1 \| M_2$. To process M using a hash function H^{H_0}, M is padded as follows: $M = M_0 \| M_1 \| \underbrace{(M_2 \| 1 \| 0^7 \| l)}_{440+72 \text{ bits}}$.

Appendix B. Message Randomization Technique RMX_{SP}

Let m be the input message, r be a message independent random bit string of size at least 128 bits and at most 1024 bits and M be the randomized message. Let $zpad$ be a

string of zero bits, which has a single zero or more 0 bits. Let λ denote zero 0 bits or an empty string. Let $pad = 1 \| zpad$. Let $rpad$ be the 16-bit binary representation of $|r|$. The input message m is encoded to the form $m \| pad$ and this encoded message is then randomized (transformed to M) as specified below:

1. If $|m| + 1 \geq |r|$:
   (a) $pad = 1 \| \lambda = 1$.
   Else
   (a) $pad = 1 \| 0^{|r|-|m|-1}$.
2. $m' = m \| pad$.
3. If $|r| > 1024$ then stop and output an error.
4. $rem = |m'| \bmod |r|$
5. Concatenate $\lfloor |m'|/|r| \rfloor$ copies of r to the rem left most bits of r to obtain R, such that $|R| = |m'|$. Now let

$$R = \underbrace{r \| r \| \ldots \| r}_{\lfloor |m'|/|r| \rfloor \text{ times}} \| r^{[rem]}$$

6. The randomized output of m is given by

$$M = \mathrm{RMX}_{\mathrm{SP}}(r, m) = r \| (m' \oplus R) \| rpad$$

B.1. *Illustration*

Let $|r| = 128$ and $|m| = 927$ bits. Now $|m| + 1 \geq |r|$, therefore $zpad = \lambda$ and $pad = 1$. Now $m' = m \| pad = m \| 1$ and $|m'| = 928$ bits. The random value $R = \underbrace{r \| \ldots \| r}_{7 \text{ times}} \| r^{[32]}$.

References

[1] S.G. Akl, On the security of compressed encodings, in *Advances in Cryptology: Proceedings of CRYPTO 83*, ed. by D. Chaum (Plenum Press, New York, 1983), pp. 209–230

[2] R. Anderson, E. Biham, Tiger: a fast new hash function, in *Fast Software Encryption*, ed. by D. Gollman. Lecture Notes in Computer Science, vol. 1039 (Springer, Berlin, 1996), pp. 89–97

[3] E. Andreeva, C. Bouillaguet, P.-A. Fouque, J.J. Hoch, J. Kelsey, A. Shamir, S. Zimmer, Second preimage attacks on dithered hash functions, in *Advances in Cryptology—EUROCRYPT 2008*, ed. by N.P. Smart. Lecture Notes in Computer Science, vol. 4965 (Springer, Berlin, 2008), pp. 270–288

[4] ANSI. ANSI X9.62:2005: Public key cryptography for the financial services industry, the elliptic curve digital signature algorithm (ECDSA) (2005)

[5] J.-P. Aumasson, L. Henzen, W. Meier, R.C.-W. Phan, SHA-3 proposal BLAKE. A finalist of NIST's SHA-3 cryptographic hash function competition, 2010. Available at http://131002.net/blake/ (Accessed on 16/08/2011)

[6] J.-P. Aumasson, W. Meier, R.C.-W. Phan, The hash function family LAKE, in *Fast Software Encryption*, ed. by K. Nyberg. Lecture Notes in Computer Science, vol. 5086 (Springer, Berlin, 2008), pp. 36–53

[7] M. Bellare, New proofs for NMAC and HMAC: security without collision-resistance, in *Advances in Cryptology-CRYPTO 2006*, ed. by C. Dwork. Lecture Notes in Computer Science, vol. 4117 (Springer, Berlin, 2006), pp. 602–619

[8] M. Bellare, R. Canetti, H. Krawczyk, Keying hash functions for message authentication, in *Advances in Cryptology—CRYPTO'96*, ed. by N. Koblitz. Lecture Notes in Computer Science, vol. 1109 (Springer, Berlin, 1996), pp. 1–15

[9] M. Bellare, P. Rogaway, Collision-resistant hashing: towards making UOWHFs practical, in *Advances in Cryptology—CRYPTO'97*, ed. by B.S. Kaliski Jr. Lecture Notes in Computer Science, vol. 1294 (Springer, Berlin, 1997), pp. 470–484

[10] S. Bellovin, E. Rescorla, Deploying a new hash algorithm, in *Proceedings of the Symposium on Network and Distributed Systems Security (NDSS)* (Internet Society, Reston, 2006). Available at http://www.isoc.org/isoc/conferences/ndss/06/proceedings/ (Accessed on 16/08/2011)

[11] R. Benadjila, O. Billet, H. Gilbert, G. Macario-Rat, T. Peyrin, M. Robshaw, Y. Seurin, SHA-3 proposal:ECHO. Second round of NIST's SHA-3 competition, 2009. Version 1.5 is available at http://crypto.rd.francetelecom.com/echo/ (Accessed on 16/08/2011)

[12] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, W. Jalby, Collisions of SHA-0 and reduced SHA-1, in *Advances in Cryptology—EUROCRYPT 2005*, ed. by R. Cramer. Lecture Notes in Computer Science, vol. 3494 (Springer, Berlin, 2005), pp. 36–57

[13] E. Biham, O. Dunkelman, A framework for iterative hash functions—HAIFA. Cryptology ePrint archive, Report 2007/278, 2007. Available at http://eprint.iacr.org/2007/278 (Accessed on 16/08/2011)

[14] E. Biham, O. Dunkelman, The SHAvite-3 hash function. A second round candidate of NIST's SHA-3 cryptographic hash function competition, 2009. Available at http://www.cs.technion.ac.il/~orrd/SHAvite-3/ (Accessed on 6/03/2011)

[15] A. Biryukov, P. Gauravaram, J. Guo, D. Khovratovich, S. Ling, K. Matusiewicz, I. Nikolic, J. Pieprzyk, H. Wang, Cryptanalysis of the LAKE hash family, in *Fast Software Encryption*, ed. by O. Dunkelman. Lecture Notes in Computer Science, vol. 5665 (Springer, Berlin, 2009), pp. 156–179

[16] J. Black, M. Cochran, MAC reforgeability, in *Fast Software Encryption*, ed. by O. Dunkelman. Lecture Notes in Computer Science, vol. 5665 (Springer, Berlin, 2009), pp. 345–362

[17] J. Black, P. Rogaway, T. Shrimpton, Black-box analysis of the Block-Cipher-based hash-function constructions from PGV, in *Advances in Cryptology—CRYPTO 2002*, ed. by M. Yung. Lecture Notes in Computer Science, vol. 2442 (Springer, Berlin, 2002), pp. 320–335

[18] F. Chabaud, A. Joux, Differential collisions in SHA-0, in *Advances in Cryptology—CRYPTO '98*, ed. by H. Krawczyk. Lecture Notes in Computer Science, vol. 1462 (Springer, Berlin, 1998), pp. 56–71

[19] S. Contini, Y.L. Yin, Forgery and partial key-recovery attacks on HMAC and NMAC using hash collisions, in *ASIACRYPT 2006*, ed. by X. Lai, K. Chen. Lecture Notes in Computer Science, vol. 4284 (Springer, Berlin, 2006), pp. 37–53

[20] I. Damgård, A design principle for hash functions, in *Advances in Cryptology—CRYPTO'89*, ed. by G. Brassard. Lecture Notes in Computer Science, vol. 435 (Springer, Berlin, 1989), pp. 416–427

[21] Q. Dang, Randomized hashing for digital signatures. NIST's special publications (800 Series). Available at http://csrc.nist.gov/publications/PubsSPs.html (Accessed on 16/08/2011), 2009

[22] Q. Dang, R. Perlner, Personal communication, October 2008

[23] D. Davies, W. Price, *Security for Computer Networks* (Wiley, New York, 1984)

[24] D.W. Davies, W.L. Price, The application of digital signatures based on public-key cryptosystems, in *Proc. Fifth Intl. Computer Communications Conference* (1980), pp. 525–530

[25] R.D. Dean, Formal aspects of mobile code security. Ph.D. thesis, Princeton University, USA, 1999

[26] B. den Boer, A. Bosselaers, An attack on the last two rounds of MD4, in *Advances in Cryptology—CRYPTO '91*, ed. by J. Feigenbaum. Lecture Notes in Computer Science, vol. 576 (Springer, Berlin, 1991), pp. 194–203

[27] B. den Boer, A. Bosselaers, Collisions for the compression function of MD5, in *Advances in Cryptology—EUROCRYPT '93*, ed. by T. Helleseth. Lecture Notes in Computer Science, vol. 765 (Springer, Berlin, 1994), pp. 293–304

[28] H. Dobbertin, Cryptanalysis of MD4, in *Fast Software Encryption*, ed. by D. Grollman. Lecture Notes in Computer Science, vol. 1039 (Springer, Berlin, 1996), pp. 53–69

[29] H. Dobbertin, Cryptanalysis of MD5 compress. Presented at the Rump Session of EUROCRYPT '96, 1996

[30] H. Dobbertin, The status of MD5 after a recent attack. *CryptoBytes* **2**(2), 1–6 (1996)

[31] H. Dobbertin, Cryptanalysis of MD4. *J. Cryptol.* **11**(4), 253–271 (1998)

[32] N. Ferguson, S. Lucks, Attacks on AURORA-512 and the double-mix Merkle–Damgaard transform. Cryptology ePrint archive, Report 2009/113, 2009. Available at http://eprint.iacr.org/2009/113 (Accessed on 16/08/2011)

[33] P.-A. Fouque, G. Leurent, P.Q. Nguyen, Full key-recovery attacks on HMAC/NMAC-MD4 and NMAC-MD5, in *Advances in Cryptology—CRYPTO 2007*, ed. by A. Menezes. Lecture Notes in Computer Science, vol. 4622 (Springer, Berlin, 2007), pp. 13–30

[34] P. Gauravaram, J. Kelsey, Linear-XOR and additive checksums don't protect Damgård–Merkle hashes from generic attacks, in *Topics in Cryptology—CT-RSA 2008*, ed. by T. Malkin. Lecture Notes in Computer Science, vol. 4964 (Springer, Berlin, 2008), pp. 36–51

[35] P. Gauravaram, J. Kelsey, L.R. Knudsen, S.S. Thomsen, On hash functions using checksums. *Int. J. Inf. Secur.* **9**(2), 137–151 (2010)

[36] P. Gauravaram, L.R. Knudsen, K. Matusiewicz, F. Mendel, C. Rechberger, M. Schläffer, S.S. Thomsen, Grøstl—a SHA-3 candidate. A finalist of NIST's SHA-3 cryptographic hash function competition, 2010. Available at http://www.groestl.info/ (Accessed on 16/08/2011)

[37] P. Gauravaram, G. Leurent, F. Mendel, M. Naya-Plasencia, T. Peyrin, C. Rechberger, M. Schläffer, Cryptanalysis of the 10-round hash and full compression function of SHAvite-3-512, in *Progress in Cryptology—AFRICACRYPT 2010*, ed. by D.J. Bernstein, T. Lange. Lecture Notes in Computer Science, vol. 6055 (Springer, Berlin, 2010), pp. 419–436

[38] P. Gauravaram, A. McCullagh, E. Dawson, Collision attacks on MD5 and SHA-1: is this the "Sword of Damocles" for electronic commerce? In *AusCERT Conference Refereed R & D Stream*, ed. by A. Clark, M. McPherson, G. Mohay (2006), pp. 1–13

[39] S. Goldwasser, S. Micali, R.L. Rivest, A digital signature scheme secure against adaptive Chosen—message attacks. *SIAM J. Comput.* **17**(2), 281–308 (1988)

[40] S. Halevi, H. Krawczyk, Strengthening digital signatures via randomized hashing, in *Advances in Cryptology—CRYPTO 2006*, ed. by C. Dwork. Lecture Notes in Computer Science, vol. 4117 (Springer, Berlin, 2006), pp. 41–59. Full version of this paper is available at http://www.ee.technion.ac.il/~hugo/rhash/rhash.pdf (Accessed on 16/08/2011)

[41] S. Halevi, H. Krawczyk, The RMX transform and digital signatures, 2006. Available at http://www.ee.technion.ac.il/~hugo/rhash/rhash-nist.pdf (Accessed on 16/08/2011)

[42] S. Halevi, H. Krawczyk, Update on randomized hashing. Technical report, 2006. Slides are available at http://csrc.nist.gov/groups/ST/hash/second_workshop.html (Accessed on 16/08/2011)

[43] S. Halevi, W. Shao, H. Krawczyk, D. Boneh, M. McIntosh, Implementing the Halevi–Krawczyk randomized hashing scheme, 2007. Available at http://www.ee.technion.ac.il/~hugo/rhash/implementation.pdf (Accessed on 19/11/2010)

[44] W. Hohl, X. Lai, T. Meier, C. Waldvogel, Security of iterated hash functions based on block ciphers, in *Advances in Cryptology—CRYPTO '93*, ed. by D.R. Stinson. Lecture Notes in Computer Science, vol. 773 (Springer, Berlin, 1993), pp. 379–390

[45] J. Jonsson, B. Kaliski, Public-key cryptography standards (PKCS) #1:RSA Cryptography specification Version 2.1. Network working group request for comments 3447, Internet Engineering Task Force (IETF), 2003. This document is available at http://www.ietf.org/rfc/rfc3447.txt (Accessed on 16/08/2011)

[46] J. Kelsey, B. Schneier, Second preimages on n-bit hash functions for much less than 2^n work, in *Advances in Cryptology—EUROCRYPT 2005*, ed. by R. Cramer. Lecture Notes in Computer Science, vol. 3494 (Springer, Berlin, 2005), pp. 474–490

[47] J. Kim, A. Biryukov, B. Preneel, S. Hong, On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1, in *Security and Cryptography for Networks*, ed. by R.D. Prisco, M. Yung. Lecture Notes in Computer Science, vol. 4116 (Springer, Berlin, 2006), pp. 242–256

[48] H. Krawczyk, Personal communication, November 2010

[49] A. Lenstra, B. de Weger, On the possibility of constructing meaningful hash collisions for public keys, in *ACISP 2005*, ed. by C. Boyd, J.M.G. Nieto. Lecture Notes in Computer Science, vol. 3574 (Springer, Berlin, 2005), pp. 267–279

[50] G. Leurent, MD4 is not one-way, in *Fast Software Encryption*, ed. by K. Nyberg. Lecture Notes in Computer Science, vol. 5086 (Springer, Berlin, 2008), pp. 412–428

[51] S. Lucks, A failure-friendly design principle for hash functions, in *Advances in Cryptology—ASIACRYPT 2005*, ed. by B. Roy. Lecture Notes in Computer Science, vol. 3788 (Springer, Berlin, 2005), pp. 474–494

[52] S. Manuel, T. Peyrin, Collisions on SHA-0 in one hour, in *Fast Software Encryption*, ed. by K. Nyberg. Lecture Notes in Computer Science, vol. 5086 (Springer, Berlin, 2008), pp. 16–35

[53] D.A. McGrew, S.R. Fluhrer, Multiple forgery attacks against message authentication codes. Cryptology ePrint Archive, Report 2005/161, 2005. Available at http://eprint.iacr.org/2005/161 (Accessed on 16/08/2011)
[54] A.J. Menezes, P.C. Van Oorschot, S.A. Vanstone, In *Handbook of Applied Cryptography*, The CRC Press Series on Discrete Mathematics and Its Applications (CRC Press, Boca Raton, 1997), pp. 321–383, Chap. 9
[55] R. Merkle, One way hash functions and DES, in *Advances in Cryptology: CRYPTO '89*, ed. by G. Brassard. Lecture Notes in Computer Science, vol. 435 (Springer, Berlin, 1989), pp. 428–446
[56] R.C. Merkle, Secrecy, authentication, and public key systems. Ph.D. thesis, Dept. of Electrical Engineering, Stanford University, USA, 1979
[57] I. Mironov, Collision-resistant no more: hash-and-sign paradigm revisited, in *Public Key Cryptography*, ed. by M. Yung, Y. Dodis, A. Kiayias, T. Malkin. Lecture Notes in Computer Science, vol. 3958 (Springer, Berlin, 2006), pp. 140–156
[58] S. Miyaguchi, K. Ohta, M. Iwata, Confirmation that some hash functions are not collision free, in *Advances in Cryptology—EUROCRYPT '90*, ed. by I.B. Damgård. Lecture Notes in Computer Science, vol. 473 (Springer, Berlin, 1990), pp. 326–343
[59] M. Naor, M. Yung, Universal one-way hash functions and their cryptographic applications, in *Proceedings of the 21st Annual ACM Symposium on Theory of Computing (STOC)* (ACM, New York, 1989), pp. 33–43
[60] National Institute of Standards and Technology. Federal information processing standard (FIPS PUB 180-3) secure hash standard, 2008. Available at http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf (Accessed on 16/08/2011)
[61] National Institute of Standards and Technology (NIST). FIPS PUB 186-2: digital signature standard (DSS). 2000. Available at http://csrc.nist.gov/publications/fips/archive/fips186-2/fips186-2.pdf (Accessed on 16/08/2011)
[62] National Institute of Standards and Technology (NIST). FIPS PUB 186-3: digital signature standard (DSS), 2009. Available at http://csrc.nist.gov/publications/fips/fips186-3/fips_186-3.pdf (Accessed on 16/08/2011)
[63] National Institute of Standards and Technology (NIST). FIPS publication 180: secure hash standard (SHS), 1993
[64] National Institute of Standards and Technology (NIST). FIPS publication 180-1: secure hash standard (SHS), 1995. Available at http://www.itl.nist.gov/fipspubs/fip180-1.htm (Accessed on 16/08/2011)
[65] National Institute of Standards and Technology (NIST). FIPS PUB 180-2: secure hash standard, 2002. Available at http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf (Accessed on 16/08/2011)
[66] NIST. Announcing request for candidate algorithm nominations for a new cryptographic hash algorithm (SHA-3) Family. Docket No: 070911510-7512-01, 2007
[67] S. Pasini, S. Vaudenay, Hash-and-sign with weak hashing made secure, in *ACISP 2007*, ed. by J. Pieprzyk, H. Ghodosi, E. Dawson. Lecture Notes in Computer Science, vol. 4586 (Springer, Berlin, 2007), pp. 338–354
[68] B. Preneel, R. Govaerts, J. Vandewalle, Hash functions based on block ciphers: a synthetic approach, in *Advances in Cryptology–CRYPTO '93*, ed. by D.R. Stinson. Lecture Notes in Computer Science, vol. 773 (Springer, Berlin, 1993), pp. 368–378
[69] C. Rechberger, V. Rijmen, On authentication with HMAC and non-random properties, in *Financial Cryptography*, ed. by S. Dietrich, R. Dhamija. Lecture Notes in Computer Science, vol. 4886 (Springer, Berlin, 2007), pp. 119–133
[70] C. Rechberger, V. Rijmen, New results on NMAC/HMAC when instantiated with popular hash functions. *J. Univers. Comput. Sci.* **14**(3), 347–376 (2008)
[71] R. Rivest, The MD4 message digest algorithm, in *Advances in Cryptology—CRYPTO'90*, ed. by A. Menezes, S.A. Vanstone. Lecture Notes in Computer Science, vol. 537 (Springer, Berlin, 1991), pp. 303–311
[72] R. Rivest, The MD5 message-digest algorithm. Internet request for comment RFC 1321, Internet engineering task force, 1992
[73] P. Rogaway, T. Shrimpton, Cryptographic hash-function basics: definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance, in *Fast Software Encryption*, ed. by B.K. Roy, W. Meier. Lecture Notes in Computer Science, vol. 3017 (Springer, Berlin, 2004), pp. 371–388

[74] R.S.A. Laboratories, PKCS #1 v2.1: RSA cryptography standard. RSA data security, Inc., 2002. Available at ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf (Accessed on 16/08/2011)

[75] Y. Sasaki, Cryptanalyses of narrow-pipe mode of operation in AURORA-512 hash function, in *Selected Areas in Cryptography*, ed. by M.J. Jacobson Jr., V. Rijmen, R. Safavi-Naini. Lecture Notes in Computer Science, vol. 5867 (Springer, Berlin, 2009), pp. 36–52

[76] Y. Sasaki, Cryptanalyses of double-mix Merkle–Damgård mode in the original version of AURORA-512. *IEICE Trans. A* **94**(1), 121–128 (2011)

[77] Y. Sasaki, K. Aoki, Finding preimages in full MD5 faster than exhaustive search, in *Advances in Cryptology—EUROCRYPT 2009*, ed. by A. Joux. Lecture Notes in Computer Science, vol. 5479 (Springer, Berlin, 2009), pp. 134–152

[78] V. Shoup, A composition theorem for universal one-way hash functions, in *Advances in Cryptology—EUROCRYPT 2000*, ed. by B. Preneel. Lecture Notes in Computer Science, vol. 1807 (Springer, Berlin, 2000), pp. 445–452

[79] M. Stevens, A.K. Lenstra, B. de Weger, Chosen-prefix collisions for MD5 and colliding X.509 certificates for different identities, in *Advances in Cryptology—EUROCRYPT 2007*, ed. by M. Naor. Lecture Notes in Computer Science, vol. 4515 (Springer, Berlin, 2007), pp. 1–22

[80] M. Stevens, A. Sotirov, J. Appelbaum, A. Lenstra, D. Molnar, D.A. Osvik, B. de Weger, Short chosen-prefix collisions for MD5 and the creation of a rogue CA certificate, in *Advances in Cryptology—CRYPTO 2009*, ed. by S. Halevi. Lecture Notes in Computer Science, vol. 5677 (Springer, Berlin, 2009), pp. 55–69

[81] X. Wang, D. Feng, X. Lai, H. Yu, Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. Cryptology ePrint archive, Report 2004/199, 2004. Available at http://eprint.iacr.org/2004/199 (Accessed on 16/08/2011)

[82] X. Wang, X. Lai, D. Feng, H. Chen, X. Yu, Cryptanalysis of the hash functions MD4 and RIPEMD, in *Advances in Cryptology—EUROCRYPT 2005*, ed. by R. Cramer. Lecture Notes in Computer Science, vol. 3494 (Springer, Berlin, 2005), pp. 1–18

[83] X. Wang, Y.L. Yin, H. Yu, Efficient collision search attacks on SHA-0, in *Advances in Cryptology—CRYPTO 2005*, ed. by V. Shoup. Lecture Notes in Computer Science, vol. 3621 (Springer, Berlin, 2005), pp. 1–16

[84] X. Wang, Y.L. Yin, H. Yu, Finding collisions in the full SHA-1, in *Advances in Cryptology—CRYPTO 2005*, ed. by V. Shoup. Lecture Notes in Computer Science, vol. 3621 (Springer, Berlin, 2005), pp. 17–36

[85] X. Wang, H. Yu, How to break MD5 and other hash functions, in *Advances in Cryptology—EUROCRYPT 2005*, ed. by R. Cramer. Lecture Notes in Computer Science, vol. 3494 (Springer, Berlin, 2005), pp. 19–35

[86] X. Wang, H. Yu, W. Wang, H. Zhang, T. Zhan, Cryptanalysis of HMAC/NMAC-MD5 and MD5-MAC, in *Advances in cryptology-EUROCRYPT 2009*, ed. by A. Joux. Lecture Notes in Computer Science, vol. 5479 (Springer, Berlin, 2009), pp. 121–133

[87] K. Yasuda, How to fill up Merkle–Damgård hash functions, in *Advances in Cryptology—ASIACRYPT 2008*, ed. by J. Pieprzyk. Lecture Notes in Computer Science, vol. 5350 (Springer, Berlin, 2008), pp. 272–289