CFRG S. Halevi (IBM)
Internet-Draft H. Krawczyk (IBM)
Expires: November 12, 2005 12 May, 2005

Strengthening Digital Signatures via Randomized Hashing
draft-irtf-cfrg-rhash-00.txt

Status of this Memo

Abstract

We propose to adopt randomized hashing as a mode of operation for existing and future cryptographic hash functions. The idea is to strengthen hash functions for use in the context of digital signatures without requiring a change to the actual hashing and signing algorithms or to their existing implementations.
We suggest that randomization can be achieved via the processing of the input to the function, even if the hash function itself is not randomized. Effective use of such mode of operation requires changes to the standardization of the encoding and processing of digital signatures (e.g., PKCS#1, FIPS186) but has no impact on existing signature and hashing algorithms. We urge the standards community to plan a transition towards these new mechanisms for which we outline specific instantiations.

1 Introduction

Recent cryptanalytical advances in the area of collision-resistant hash functions (CRHF), specifically the attacks against MD5 and SHA-1, have not only shaken our confidence in the security of specific constructions but, more fundamentally, have cast doubts on our ability to design collision-resistant hash functions that will withstand attacks over a long period of time. These attacks remind us that cryptography is founded on heuristic constructions whose security may be endangered unexpectedly. In particular, this highlights the importance of following two fundamental cryptographic design principles:

(i) design protocols and applications such that the underlying cryptographic pieces (e.g., hash functions) are easy to replace when need arises (in particular, avoid hard-wiring of any specific construction into the application), and
(ii) design as general as possible mechanisms with as little as possible requirements from the basic cryptographic building blocks.

The present proposal is intended to address these points, especially the second one.

Although many existing applications that use hash functions do not actually require full collision resistance, and although the current attack on SHA-1 is not quite practical yet, it is clear that we cannot dismiss the recent attacks as theoretical only. Indeed there are important applications today that do rely on full collision resistance, in particular those that use standard signature schemes to provide non-repudiation or certification services. And with the expected cryptanalytical improvements in the near future, ignoring these new attacks would be irresponsible. Some of the options contemplated in the applied cryptography world for responding to the recent attacks on MD5 and SHA-1 are the following:

(1) Modify applications that rely on collision resistance such that the particular use of CRHF in these applications will be less vulnerable to collision attacks.

(2) Upgrade the systems using SHA-1 and MD5 to use stronger hash functions such as the SHA2 family (256- and 512-bit versions). The hope is that these functions will provide for more robust CRHFs.

Option (1) could be applied to different settings, but it is very application specific. In particular, note that even if one could set precise assumptions on the way specific applications are used today, these assumptions are likely to change or become obsolete over time. To illustrate this point, consider modifying applications that use signatures so that the messages to be signed are structured in a way

that is unpredictable to the attacker. This approach relies heavily on the understanding of the semantics and structure of messages used in the application. Therefore, while it may be viable for specific applications (such as choosing unpredictable serial numbers in

certificates) it is insufficient as a general measure.

Option (2) is more robust but its cost and complexity are significant: it requires a multitude of applications, protocols and implementations to instrument the transition to new functions. Hopefully, the current attacks will improve at a mild enough pace so that a relatively orderly transition can be implemented. And even if we manage such a gradual transition, we must contemplate the possibility that by the time the transition is completed the new adopted functions will appear as weak as SHA-1 appears to be now.

The approach that we propose here takes elements from the above two options. We suggest that we must plan for a transition to more secure mechanisms, that this has to be done in an orderly way (i.e., not as an uncontrolled panicking reaction), and that rather than patching individual applications we re-engineer general mechanisms in a way that provides for more robust cryptography, specifically more secure signature schemes. To accomplish the latter we propose to re-define the way hash functions are used in the context of digital signatures so as not to rely so heavily on the full collision resistance of our hash functions. This is likely to result in a significantly longer useful life for SHA-1 itself and, even more importantly, will result in significantly weaker requirements from any hash family to be adopted or designed in the future.

While this solution is not for free (see below), we show that it can be done without having to change the basic signature algorithms in use (e.g., RSA, DSS), without even changing the existing hash functions (e.g., SHA-1, SHA2), and without the need to understand the semantics of particular applications or messages. What needs to be changed is the interface to the signing and hash algorithms. The main tool for achieving all of the above, in particular lowering the requirements on the security of hash functions, is the use of randomized hashing, a well-studied notion in the cryptographic literature (but seldom used in practice). Since our proposal requires no change to the hashing algorithms themselves it can be seen as a proposal for a "mode of operation" for existing and future hash functions.

We end this introduction by noting that randomized hashing has applications beyond the context of digital signatures. On the other hand, it is important to also realize that randomized hashing is NOT a replacement for CRHF in ALL possible applications. For example, randomized hashing may not be appropriate in applications where commitment is required or implied, e.g., a bidder committing to her bid in an auction. So while we do not advocate abandoning CRHF as a useful cryptographic tool, the important message we wish to convey is that having a randomized mode of operation for CRHF for use in

digital signature applications, such as those requiring non-repudiation, provides a substantial security gain and significantly raises the bar against existing and future cryptanalytical attacks on the underlying hash functions.

2. Randomized Hashing and Signature Schemes

The idea behind the use of randomized hashing is simple. Instead of using a single deterministic hash function such as SHA-1 one uses a keyed hash function (equivalently, a family of hash functions where each function in the family is determined by a key or index). For example, SHA-1 itself can be converted into a family of hash functions indexed through a variable IV (we mention this as an illustration, not necessarily as the best way to transform SHA-1 into an indexed hash family for our purposes here).

Let us denote by SIG a secure signing algorithm (such as RSA or DSS), by H a family of hash functions, and by H_r the function from this family indexed by the value r. Now, for signing a message m the signer chooses a random value r and computes SIG(r,H_r(m)). Here, the pair (r,H_r(m)) represents a (standard) encoding of the concatenation of the values r and H_r(m). The signature on message m now consists of the pair (r,SIG(r,H_r(m)). Before discussing implementation issues (such as the choice of the family H, the index r, and the encoding function) let's see why this method reduces the reliance on collision resistance of the hash function.

Consider an attacker, Alice, against a honest signer Bob that signs using the scheme from above. Alice provides a message m to be signed by Bob and she gets back the pair (r,SIG(r,H_r(m)) from Bob, where r is a value chosen at random (or pseudo-randomly) by Bob anew with each signature. How can Alice attack this scheme (short of breaking the signature algorithm, say RSA, itself)?

What Alice needs to do is to find a message m that Bob is willing to sign and hope that when she receives the pair (r,H_r(m)), for random r chosen by Bob, she will be able to find another message m' for which H_r(m)=H_r(m') (with the same index r chosen and signed by Bob). If she could do that then the signature string SIG(r,H_r(m)) would also be a valid signature for m'.

We remark that Alice could do a bit better by asking to sign many messages m1, m2,..., getting back many pairs (r1, SIG(r1,H_r1(m1)), (r2,SIG(r2, H_r2(m2)),..., and then finding another m' such that for some i it holds that H_ri(mi)=H_ri(m'). But note that the number of pairs is limited by the number of signatures that Bob is willing to generate, and that Alice needs to engage in an on-line interaction with Bob for every such pair. It is therefore likely that in most applications the number of pairs available to Alice would be quite small (say, not more than 2^30 or 2^40). Below we analyze only the case of a single pair, while keeping in mind this additional factor when dealing with concrete parameters.

Returning to the single pair condition, we see that Alice can produce a forged signature if she can do one of the following:

(i) Cryptanalyze the family H to the point that for a random pair (r,v) she can find m' such that H_r(m')=v.
(ii) Achieve (i) when in addition to the pair (r,v) Alice also knows another value m for which H_r(m)=v.
(iii) Achieve (ii) when the value m is chosen by Alice herself BEFORE learning r.

In other words, the collision finding task for Alice is not against a fixed, known in advance, function as it is the case today with the use of a fixed hash function, but against a random function in the hash family H whose index r is revealed to Alice only after she committed to the message m. In particular, being able to find collisions against a fixed member of the family is useless; Alice needs to be able to do so for a reasonably large fraction of hash functions in the family.

Before we continue we note that the resistance to each of the above forms of attacks is called, respectively:
(i) one-wayness (OW)
(ii) second-preimage resistance (SPR)
(iii) target-collision resistant (TCR)

The precise difference between SPR and TCR is that in the former the first message m is chosen at random while in TCR the attacker gets to choose m (but before learning r). We also remark that TCR functions were first defined by Naor and Yung [NY89] where they were called universal one-way hash function (UOWHF); the term TCR that we use here is from [BR97].

Obviously, these tasks are harder to perform than a regular collision-finding attack against a single CRHF function H (i.e. the finding of two messages m,m' such that H(m)=H(m')).
More specifically, one can point to two essential differences between a regular collision attack and any one of the above tasks. First, a regular collision attack can be performed in a complete off-line manner (i.e. ahead of the time when a signature is to be issued) while each of (i)-(iii) depends on the choice of r and therefore needs to be completed only after r is determined and communicated to the adversary. Second, while collisions against a single hash function that outputs k bits can be found by brute force in time 2^{k/2}, a brute force TCR attack will take 2^k time.
And even if we recall the additional factor of 2^n pairs that Alice can achieve via on-line interaction with Bob, a brute force attack would still take her 2^{k-n} time (in the case of SHA-1 k=160, while n would be no more than 40 in most reasonable applications).

Of course, none of the above says that SHA-1 (or any other specific hash function) is sure to resist TCR attacks (or even SPR attacks). But it clearly indicates that if an application uses a hash function

in a way that can only be broken under a successful TCR attack, then the application is much more likely to remain secure in face of cryptanalytical improvements than one that relies on full collision resistance. This is true whether the hash function in use is a

partially broken CRHF such as SHA-1, a (hopefully) better collision-resistant family such as SHA2, or any hash function to be designed in the future.

While the above indicates general relations between the strengths and vulnerabilities of different hashing tasks it does not tell us how to instantiate a TCR function. We discuss this in the next section. Later, in section 4, we explain how to integrate randomized hashing into signatures (specifically, how to sign and transport the index r).

3. A TCR Construction for Iterated Hash Functions

We propose a specific way to convert a single hash function H (e.g SHA-1 or SHA2) into a TCR function family. The design principles that we follow are:

(1) Do not change H: randomization is applied to the hash input before the hash function is called.
(2) Minimize performance impact.
(3) Increase (heuristically) the likelihood of resistance of the family to TCR attacks.

In 3.1 we present a basic construction (with some heuristic rationale in Appendix A). In 3.2 we list some variants which take into account some further trade-offs between performance and plausible security. We stress that these methods, although plausible, need to be scrutinized further before they can be adopted.

3.1 A simple randomized hash construction

Let H be a hash function that processes the message to be hashed in 512-bit blocks. For example, if H is an integrated hash function a-la-Merkle-Damgard then the underlying compression function has as inputs an IV and a 512-bit data input. (We use 512 bits as the typical block size but other values are possible.) Let XOR denote the bit-wise exclusive-or operation.

Given a message m to be hashed, the signer (or "hasher") chooses a 512-bit random value r, and XORs each 512-bit block of m with r. (If m is not an exact multiple of 512-bit blocks then the shorter last block is XORed with an appropriately truncated r.)
In other words, we concatenate r to itself until we get a string r* of the same length of m, and then compute m XOR r*.
We define H_r(m) to be H(m XOR r*).

Note: By our definition the result of (m XOR r*) is of the same length as m; therefore, the length padding defined by Merkle-Damgard

functions such as SHA-1 is applied to (m XOR r*). In other words, the length padding is not subject to the XOR with r*.

In Appendix A we provide some rationale on the choice of this particular way of converting iterated hash functions into TCR. Variants of this method are presented next.

3.2 Some Randomized Hash Variants

A possible strengthening of our construction from Sec 3.1 can be obtained if, in addition to XORing each block of input with the value r, one also prepends r to the input to H, i.e., the input to H consists of the block r concatenated with (m XOR r*). This provides a randomizing effect to the initial IV of H (in the spirit of the HMAC construction).

An even more conservative variant could interleave the block r between any two blocks of the original message, thus providing an IV randomization feature for each application of the compression function. The obvious drawback is the added computation (double the cost of the original hash function).

Another natural idea is to add a layer of security by XORing a different random pad to each block of the message. Clearly, this adds a non-trivial computational cost (one would need to generate a pad of the length of the message via some PRG). A midway strategy could be to start with a pad of the length of a single block and slightly (and inexpensively) change this pad for each new block of input, for example by applying circular byte rotation to the previous block pad. A similar idea would be to derive the pad from a byte-oriented LFSR whose initial value is the key r.

Finally, if the generation of a 512-bit random (or pseudo-random) quantity r for each signature is regarded as expensive (possibly true for low-power devices, smart cards, etc.) then it is possible to define r as the concatenation of a shorter pad. For example, in order to define r one could choose a random 128-bit string and concatenate it four times to create r. Given the heuristic nature of our constructions this may be considered a reasonable trade-off.

4. TCR Hashing and Signature Encoding

Recall how randomized hashing is to be used in the context of digital signatures. For signing a message m, the signer chooses at random a value r and computes SIG(r,H_r(m)) where SIG represents a signing algorithm (such as RSA or DSS). More precisely, the signer will use a well-defined standard encoding of the concatenation of the values r and H_r(m) and then apply algorithm SIG to this encoding.
The signature on message m consists of the pair (r,SIG(r,H_r(m)).

The above requires changing current signature schemes in four ways:

(1) Choosing a random (unpredictable) index r for each signature,

(2) Replacing the current hashing of a message m from H(m) to H_r(m),
(3) Signing r, and
(4) Transporting r as part of the signature.

Here we discuss the required changes to existing message encodings for implementing the last 3 points. We focus on the two main algorithms in use: RSA and DSS. We note that while changing existing encoding standards may be one of the possible obstacles to adopting randomized hashing, this change is instrumental in allowing for more secure and robust signature schemes not only in the short term but in the farther future as well. We suggest that this change to the standards be specified and adopted as soon as possible. As we see below, these changes can be specified in a way that is independent of the specific randomized hash function to be used.

We start with RSA. The most common encoding in use with RSA signatures is PKCS#1 v1.5. It specifies that given a message m to be signed, the input to the RSA signature function is a string composed of the hash value H(m) (computed on the message m using a deterministic hash function such as SHA-1) which is padded to the length of the RSA modulus with a standard deterministic padding (this padding contains information to identify the hash algorithm in use). This encoding can be extended to deal with randomized hashing as follows. First, the value H(m) is replaced with H_r(m) for r chosen by the signer. Second, part of the deterministic padding (which is currently filled with repeated 0xff octets) is replaced with the value of r. In this way, r is signed and, at the same time, it is made available to the verifier of the signature without any increase in the size of signatures (r is recovered by the verifier by inverting the signature operation).

Another RSA encoding, called EMSA-PSS, is standardized by PKCS#1 v2.1 and is based on the randomized signature scheme of Bellare and Rogaway [BR96]. Unfortunately, the standard defines an encoding in which the first step is to apply a deterministic hash function (say, SHA-1) to the message m. Only then the randomized encoding scheme of PSS is applied. As a result, the signature scheme that uses EMSA-PSS is broken if the hash function is not fully collision resistant. In order to use this scheme with randomized hashing, one would replace the current H(m) value in the encoding with H_r(m) and the value r would be encoded in a way that the verifier of a signature can recover it before applying the randomized hashing. The original PSS scheme from [BR96] can be used, or adapted, to achieve such an encoding.

Two points to remark regarding the applicability of PSS here are: first, the original PSS scheme is patented -- see US Patent 6266771 (which may or may not be an obstacle for adoption). Second, the main analytical benefit of PSS is its provability based on the so called "random oracle model". While this provides a good heuristic backing to the construction, one has to take into account that here we are dealing explicitly with lowering the security requirements from the hash function, so it is questionable how random-like these

functions may be required to be. Formal proofs aside, the PSS scheme offers good heuristic advantages over the PKCS#1 v1.5 in that it better randomizes the input to the RSA signing algorithm.

Regarding the DSS (or DSA) signature algorithm, the first thing to note is that this is already a randomized signature scheme. A DSS signature is composed of a pair of elements (R,S) where R is a random element in the DSS group and S is a value computed as a function of the private key of the signer, the discrete logarithm of R (denoted k), and the value H(m) (where m is the message to be signed and H a deterministic hash function). In order to convert this scheme to use randomized hashing one can use R itself as the index to the hash family, i.e., r=R (or to derive r from R in some deterministic way). Then one would replace H(m) with H_r(m). In this way the size of signatures is unchanged and no further processing is required to generate r. Also note that while the signature component R is not strictly "signed", the attacker cannot control or choose this value (indeed, an attack that finds values R and S for which (R,S) are a valid signature of H_R(m), for a value H_R(m) not signed by the legitimate signer, would contradict the basic security of DSS). One may argue that the use of H_r(m) instead of H(m) can be viewed as an "implementation" of the random-oracle version of DSS as analyzed by Pointcheval and Stern [PS96]; the same caveats expressed in the case of PSS in relation to the use of the random oracle model apply here as well.

One consideration in regards to using the component R of DSS signatures as the index to the randomized hash family is that, in order to ensure the TCR property, this index needs to be unknown (unpredictable) to the attacker when the latter chooses the message m to be signed. If the value of R is computed off-line by the signer (which is possible in the case of DSS) and is leaked before the attacker choses m then the benefit of randomized hashing is lost. Hence, R=g^k should be kept secret together with k until the signature is issued. This is not a fundamental limitation to the practice of DSS since the DSS scheme already requires (in an essential way) that k be kept secret, even if computed off-line, since its discovery by the attacker is equivalent to finding the secret private key of the signer!

5. Security Considerations

This document presents mechanisms that, if adopted by standard bodies such as the IETF, will result in significant improvements to our current and future digital signature systems. While this document focuses on randomized modes of operation of hash functions that provide randomized hashing without changing existing algorithms, it is advisable that future hash families will be designed with randomized hashing and TCR requirements in mind. For example, new schemes that follow the Merkle-Damgard approach may consider allowing for the masking of intermediate values with optional user provided inputs (that is, such a mask could be set to

a default value, say 0, for deterministic uses of the hash function, and to user-provided values when randomization is desired). The important point is that implementations of the function will be ready to accept such masks without having to change the function.

We note that all references to "randomness" in this document should be interpreted as "pseudo-randomness" provided one uses a cryptographically strong pseudorandom generator (or pseudo-random function) initialized with a strong unpredictable seed.

We also mention that using TCR hashing may mean that the legitimate signer can find two messages with the same signature (since it is the legitimate signer that is choosing the randomness r). One should note, however, that this has no bearing on non-repudiation (as the signer is still bound to both messages). Moreover, as shown in [SPMS02], even if one uses CRHF some secure signature schemes (such as ECDSA) may allow a signer to find two different messages whose signature string is the same. Still, as mentioned at the end of the Introduction, there may be OTHER applications of CRHF that cannot be replaced with a TCR family.

Finally, the general approaches to randomized hashing and digital signatures discussed here do not depend on the specifics of the concrete constructions that we proposed here. Other forms of randomized hashing and TCR schemes may be superior to the ones proposed here and further proposals are encouraged.

ACKNOWLEDGMENT. We thank Ran Canetti for useful discussions and for badgering us to write this document.

REFERENCES

[BR96] M. Bellare and P. Rogaway, "The Exact Security of Digital Signatures -- How to Sign with RSA and Rabin", Eurocrypt'96, LNCS 1070, 1996.

[BR97] M. Bellare and P. Rogaway, "Collision-Resistant Hashing: Towards Making UOWHFs Practical", Crypto'97, LNCS 1294, 1997

[HPL04] D. Hong, B. Preneel, and S. Lee, "Higher Order Universal One-Way Hash Functions", Asiacrypt'04, LNCS 3329, 2004.

[NY89] M. Naor and M. Yung, "Universal One-Way Hash Functions and Their Cryptographic Applications", STOC'89, 1989.

[PS96] D. Pointcheval and J. Stern, "Security Arguments for Digital Signatures and Blind Signatures", J.Cryptology, 13:361-396, 2000.

[S00] V. Shoup, "A Composite Theorem for Universal One-Way Hash Functions", Eurocrypt'00, LNCS 1807, 2000.

[SPMS02] Jacques Stern, David Pointcheval, John Malone-Lee, and

Nigel P. Smart, "Flaws in Applying Proof Methodologies to
Signature Schemes", CRYPTO '2002, LNCS 2442, 2002.

Appendix A - Rationale for the Proposed TCR Construction(s)

Our TCR proposals follow the following principles:

(1) Allow the use of existing functions such as SHA-1 and SHA2 (in particular, iterated hash functions a la Merkle-Damgard).
(2) Do not change the hash function but only the interface to it (e.g., in our proposal randomization is achieved via the input to the function and therefore implemented hash functions, in either software or hardware, can be used without modification).
(3) Use as weak as possible properties of the compression function underlying the hash construction.

Our construction is general enough to be used with any hash function that processes the incoming data as blocks. Yet, we focus in our discussion here on Merkle-Damgard (M-D) type of hash functions since these are the most common schemes in practice.

While (1) and (2) are obvious properties of our suggested construction we elaborate here on point (3). Ideally, we would have liked to provide a mathematical theorem proving the security of our construction using only relatively weak requirements from the underlying compression function. While such theorems exist for some specific constructions (e.g., [BR97,S00]), they all include operations that violate the principle of using the existing hash functions without any change (e.g., masking the intermediate value of the compression function with each call to this function). We thus settle for a heuristic rationale that should be scrutinized in light of the evolving ideas in the area of hash function cryptanalysis.

Let H be a M-D function (the reader can think of SHA-1 for concreteness) and h be the corresponding compression function. That is, h acts on two inputs, a and b, where a represents an intermediate value (IV) and b is a 512-bit block, and the output of h is of the length of the IV (IV lengths vary with different constructions, e.g., 160, 256, etc.). The function H itself is defined for arbitrary inputs by iterating h over the successive blocks of the input with each iteration using the IV computed by the previous application of h (the first IV is set to some constant defined by the specification of H).

Consider now a family of compression functions derived from h as follows: for each 512-bit index r, define h_r(a,b)=h(a,b XOR r). It is easy to see that iterating h_r as in a M-D construction one obtains the function H_r that we defined in 3.1.

Merkle and Damgard showed that if a compression function h is collision resistant with respect to fixed-length inputs, then the function H obtained by iterating h is collision resistant on

arbitrary inputs. We would like to claim the same with respect to the property of target-collision resistance (TCR), namely, that if h is TCR so is H. This, however, is not necessarily the case.

Yet, an "approximation" to this result was recently shown by Hong, Preneel and Lee [HPL04]. They show that if the construction h_r has a property called "n-order TCR" then the iterated family H_r is TCR for messages of up to n blocks. The property of n-order TCR is defined by the following game between Alice (the Attacker) and a "hasher" Bob.

(1) Bob chooses an index r and keeps it secret.
(2) For i=1,...,n: Alice chooses a pair (a_i,b_i) and receives from Bob the value h_r(a_i,b_i).
(3) Alice chooses a pair (a,b).
(4) Bob reveals r to Alice

Alice wins the game if she can find (a',b') different from (a,b) such that h_r(a,b)=h_r(a',b').

In other words, Alice needs to carry a TCR attack but she is allowed to query h_r on n inputs of her choice before committing to the first colliding message and before learning the value of r. Intuitively, the difference with a regular TCR attack is that Alice has now an advantage in choosing (a,b) since she can first learn something about r from the first n queries.

A family h is called n-order TCR if any efficient attacker (Alice) can only find (a',b') as above with insignificant probability. Before we continue it is important to clarify that the above game defining n-order TCR functions is not a game that reflects an actual interaction between an attacker and a victim in real life but it is only a virtual game used to define the security of a function.

How much does the extra phase (2) in the game from above help Alice to find collisions? This of course depends on the specific function, and to some extent also on the value of n. Note that if one lets n to be huge (say 2^80 in the case of SHA-1) then Alice can use this "learning phase" to find colliding pair (a_i,b_i) and (a_j,b_j) that she can then use as (a,b) and (a',b') respectively. But recall that n represents the length in blocks of the messages to be hashed with the iterated construction, so it will typically be quite small. (I.e., n=4 or so in the case of certificates, and n < 2^30 even for huge documents.) Hence one may hope that the learning phase will not be sufficiently useful for Alice to choose the colliding pair.

In other words, while in order to break a collision-resistant hash function an attacker can spend a HUGE amount of OFF-LINE computation for finding collisions, for breaking an n-order TCR function the attacker is limited to only MODERATE ON-LINE interaction with the hasher after which it needs to commit to a first colliding value x. Only then the attacker receives the actual value r for which it needs to find x' such that h_r(x)=h_r(x').

We also comment that the common view of the compression function h(a,b) as a block cipher with key b and input a gives rise to another heuristic argument supporting the view of h_r as n-order TCR. Viewing h(a,b) as a block cipher, phase (2) of the attack from above

on h_r is just a chosen-plaintext related-key attack on the block cipher h. If h resists such attacks with a moderate number of queries, then phase (2) does not help the attacker learn much about r. Hence, if h is both TCR and a sufficiently robust block cipher, then it is also an n-order TCR.

As said, [HPL04] show that if a compression family h={h_r} is n-order TCR then the family H={H_r} is TCR on n block inputs (here H_r is a Merkle-Damgard iteration of h_r). Applying this result to our case, we obtain that if the construction h_r(a,b) = h(a, b XOR r) is an n-order TCR then the family H_r described in 3.1 is TCR for n-block inputs.
In other words, any TCR attack against the family H_r that uses n-block messages, translates into an n-order TCR attack against the compression function family h_r with only n initial oracle queries.

This provides some foundation to the belief that even the existing hash functions are significantly more secure in the sense of TCR than for collision resistance when used as specified here.
In addition, one should examine the current attacks and see to what extent they apply to the defined functions. In particular, we note that the XORing of input blocks with a random block, while it preserves differentials, it also destroys the ability of the attacker to set some of the bits of the colliding messages to values of its choice. It seems that an attack that takes advantage of differentials in this setting would need to rely on universal differentials that depend only on the hash function and for which most pairs of messages with that difference would collide.

Finally, we point out to another "motivating" element in our design. Remember that SPR (second pre-image resistant) functions are a weaker (i.e., easier to accomplish) version of TCR functions where the attacker cannot choose the first colliding value but rather this value is determined at random. A straightforward way to transform an SPR compression function h into a TCR family [S00] is to choose a pair r=(s1,s2), where s1,s2 are random values of the length of the IV and block size, respectively, and define h_r(a,b)=h(a XOR s1, b XOR s2). Unfortunately, iterating such an h_r is impractical as it requires modifying H such that the IV can be XORed with S1 in each iteration of h. Therefore, instead of using this full transformation of SPR into TCR we carry the randomization only in the second input of h, namely, in our construction in 3.1 we use h_r(a,b)=h(a,b XOR r) (when viewing h as a block cipher, as mentioned before, the XORing with r provides for randomization of the cipher key).

Authors' Addresses

Shai Halevi
shaih@alum.mit.edu
Hugo Krawczyk
hugo@ee.technion.ac.il

IBM T.J. Watson Research Center
19 Skyline Drive
Hawthorne, NY 10532
USA

Full Copyright Statement