The Encrypted Elliptic Curve Hash

Daniel R. L. Brown*

February 6, 2008

Abstract

Bellare and Micciancio's MuHASH applies a pre-existing hash function to map indexed message blocks into a secure group. The resulting hash is the product. Bellare and Micciancio proved, in the random oracle model, that MuHASH is collision-resistant if the group's discrete logarithm problem is infeasible. MuHASH, however, relies on a pre-existing hash being collision resistant. In this paper, we remove such a reliance by replacing the pre-existing hash with a block cipher under a fixed key. We adapt Bellare and Micciancio's collision-resistance proof to the ideal cipher model. Preimage resistance requires us to add a further modification.

Key Words: Hash Function, Collision Resistance

1 Introduction

The MuHASH construction of Bellare and Micciancio [BM97] provides a hash function that is provably collision resistant under plausible assumptions. That is to say, Bellare and Micciancio identified a set of conditions sufficient to make MuHASH collision resistant. In this light, MuHASH provides excellent security.

On the other hand, MuHASH uses a pre-existing hash function operating over a fixed length. This pre-existing hash function must be both collision resistant and one-way in order for MuHASH to be collision-resistant and one way, respectively. In other words, a necessary condition for MuHASH to be a secure hash is that some pre-existing hash is secure. In this light, MuHASH provides mostly a way to extend the domain of an already secure hash. In particular, it does not produce collision resistance from scratch.

It would be desirable to modify MuHASH in such a way to reduce the reliance upon some pre-existing secure hash. More formally, we would like to weaken the set of assumptions *necessary* for security without (significantly) strengthening the set of assumptions *sufficient* for security. More generally, we would like to build a collision resistant function from scratch.

This paper introduces a minor modification of MuHASH, called Encrypted Elliptic Curve Hash (EECH). The main change from MuHASH is to use a secure block cipher, with a fixed non-secret key, instead of a pre-existing hash function.[1] Because a block cipher has no collisions, EECH does not directly rely on some conjectured collision resistance of a pre-existing hash function. Therefore, in EECH, we have weakened the set of conditions necessary for collision resistance.

*Certicom Research

[1] Arguably, the description of MuHASH [BM97] can be interpreted to include this case, since this function is called a *randomizer* or *compression* function, with the former being open to interpretation as an encryption function. Under this interpretation, then, MuHASH, like EECH is no longer one-way.

The proof in the random oracle model of collision resistance of MuHASH [BM97] is adapted here to work for EECH by using the ideal cipher model instead of the ideal hash (random oracle) model. We take the position that this change does not substantially strengthen the set of assumptions sufficient for the collision resistance.

On the other hand, the use of a block cipher in EECH opens the door to preimage attacks (single block only). To close this door, we propose various enhancements of EECH by introducing a further stage of processing. Again, we wish to close this door without relying on some pre-existing preimage (or collision) resistance assumption.

Admittedly, the ideal cipher model is a rather strong assumption. Although we have used this strong tool as a crutch to find conditions sufficient for security, we speculate, however, that a secure cipher is not actually necessary for the security of EECH (and its preimage resistant enhancements). As a logical test for this conjecture, we propose a simplification of EECH where the block cipher is replace by the identity function. We call this the Elliptic Curve Only Hash (ECOH). The one-way enhancements of EECH are defined for ECOH analogously.

2 Encrypted Elliptic Curve Hash

A message M will be divided into blocks $M_1, \ldots, M_m$, where the last block M_m is reserved to encode the length of M, and furthermore has formatting distinguishable from any other block. The *pre-hash* is computed as

$$G = G(M) = \sum_{j=1}^{m} E_k(j \| M_j) \tag{1}$$

where E_k is a secure block cipher, but with a fixed and non-secret key k. More precisely, we may actually have the terms in the form $E_k(c \| j \| M_j)$ where c is the minimal value of counter that ensures the output represents a valid element in some abelian group $\mathbb{G}$, whose operation we have indicated using addition (whereas MuHASH [BM97] uses multiplicative notation). We assume throughout this paper that $|\mathbb{G}| = n$ is prime and that $\mathbb{G}$ is the subgroup of an elliptic curve group defined over a finite field.

We will call the function computing G the Encrypted Elliptic Curve Hash (EECH). We note that, like MuHASH, this function is *incremental*. Suppose that message M' is identical to M except in block j, where it has value M'_j instead of M_j. We can compute $G(M')$ knowing only $G(M)$ and M_j and M'_j as

$$G(M') = G(M) - E_k(j \| M_j) + E_k(j \| M'_j) \tag{2}$$

For long messages, this will be considerably faster than computing $G(M')$ using the full summation in (1).

We also say that EECH and MuHASH are δ-*incremental*, because, not only is (2) faster than (1), but it only requires the knowledge of the changes (the δ). Therefore, its cost does not depend on the length of the M nor does it require full knowledge of M.

A special case of incrementality is modification of the final blocks, including deletion of the final blocks, or addition of new blocks. Iterative hash functions, such as SHA-1 and SHA-256, are not generally incremental, but they often allow efficient addition of new blocks (but the length-encoding cannot be removed, so the extended messages will have peculiar message encodings embedded within them). One can also consider a more general kind of incrementality that allows for more than just in-place changes. For example, updating the hash after insertions and deletions of block into the middle of message. We will not directly address such general incrementality. We only comment

that it can be done with techniques such as hash trees or pointers, and it requires more complex data structures than bit strings.

Although δ-incrementality can be viewed as a positive attribute, one can also view it negatively, and re-name it *malleability*. For example, because of the extension properties of SHA-1, one could not define a MAC by prepending a message with a secret key and applying SHA-1, because then an attacker could produce the MAC of any extension. Similarly, if a MAC were to be built from EECH by inserting a secret key into the message, the δ-incrementality would allows an attack to forge a MAC of the modification of the message. We will not take this rather negative view, but simply warn the reader not to construct a MAC in this way. A better way of constructing a MAC from EECH would be to make the block cipher key k secret, but even this is probability not the best way to create a MAC, so we will not delve further here into the question of MAC design.

We will argue that G is suitable specifically for use as a collision-resistant hash value. Even so, it is not suitable for a general purpose hash because it lacks complete preimage resistance, which is commonly expected from a hash function. Specifically, an attacker who knows all but one of the message blocks, say M_j, can deduce the M_j from G by first subtracting the contribution of the remaining known message blocks, then inverting E_k. Therefore we provide various enhancements of EECH to eliminate this preimage attack.[2]

2.1 Fitting into the Group

For an elliptic curve group over a finite field F, approximately half of the field elements f are equal to the x-coordinate of a point P on the elliptic curve over the field. With the approach described earlier, of finding the minimal value of an unsigned integer counter c such that $E_k(c\|j\|M_j)$ encodes a valid x coordinate, the expected average value of c will be 1, with value 0 occurring half the time. Occasionally, c will be need to higher, and it is arguably costly to do so many encryptions and elliptic curve validity testing. Probably more problematic is that if only u bits are allotted for the c, then ther is probability 2^{-u} that no encryption corresponds to a valid x-coordinate. This forces us to choose u fairly large, to avoid failure to evaluate the hash. If we want to resist an mischievous denial-of-service attack, then we need to choose u even larger to make it infeasible for an adversary from search for a message block M_j which will never encrypt to a valid x-coordinate.

Rather than choosing large u, we could keep u small, except that when we find that no values of c lead to a valid encryption, we instead subdivide M_j into two sub-blocks and try to encrypt each of these separately, perhaps new counters with a larger value of u. This might make it harder for adversry to find an unhashable message, but might make it easier for an adversary to produce a message that takes twice as long as normal to hash.

A different strategy is to use quadratic twists of curves. Generally, if a value $x \in F$ is such that x is not valid for the ellitpic curve, then it is valid for the twist of an elliptic curve. Therefore, one could add the valid points on the main elliptic curve, and add the invalid points on the twist of the elliptic curve. This gives two points. One could either concatenate these two points, or more in the spirit of EECH, add them, as follows. To add a point on the curve to a point on the twist, we map the point on the twist into an point on the main curve extended to a quadratic extension of the field F. Either way, concatenation or using a quadratic extension, the result is twice as long.

It should be noted here, that the y-coordinate has not been mentioned. There is negligible chance that the output of an encryption operation will encode both an x-coordinate and a y-coordinate. Therefore, we aim for the x-coordinate, and perhaps one bit of the y-coordinate. Then

[2]We may use the term prEECH to indicate one of the enhancements.

we can use standard methods of point decompression to recover the y-coordinate, which we need in order to do the point additions.

It should be noted that this point decompression involves solving a quadratic equation, which can be a costly operation compared to an encryption or a elliptic curve point addition. For curves defined over characteristic two fields, the quadratic equations can be solved using the half-trace function, which is linear. With the use of look-up tables and linearity, it may be possible to make this cost relatively small. Decompression over a binary field also requires the computation of an inverse, which can also be somewhat costly. However, the cost inversions can be amortized by batching them together.

2.2 Minimalism

To make EECH preimage resistant, the most natural idea is to apply a one-way function, such as a hash function. In doing so, however, we would not like to introduce some necessary conditions, specifically collision resistance and preimage resistance of the applied hash function. We wish to be *minimalist* in our modification of EECH, with this taking the three tenets: (a) no new necessities, (b) translation of collisions, (c) cipherless security. These tenets are explained further below.

Consider the hash function derived from EECH defined as follows

$$H = F_k(G) = G \oplus E_k(G). \tag{3}$$

Although the function F_k above is likely be an excellent choice[3] of preimage and collision resistant function, along with having good pseudorandomness properties, it does not meet any of our three minimalist tenets.

If F_k is not preimage resistant, then given H, an adversary has a good chance of finding G (because F_k is almost one-to-one). As stated earlier, G has the property if an adversary knows all but one block of the message, then the adversary can find the missing block given G. Therefore H has the same property. Therefore, for H to have full preimage resistance, it must be the case that F_k is preimage resistant.[4] In other words, (3) adds a new necessity: preimage resistance of F_k, failing to meet the first of the three tenets. Arguably, however, the preimage resistance is fairly well-accepted assumption, so it is not really new. We will take the subjective perspective that the security EECH is based on the security of an elliptic curve group, so that this necessity is new with respect to the design rationale for EECH in that it does not depend on the elliptic curve group.

Suppose that one can find collisions in the function H defined by (3). It is not clear how one translates this into collisions in the function of G (which is EECH). Therefore, H also fails to meet the second of our three tenets. Perhaps in the ideal cipher model, one can make such a translation, but we do not want to invoke a non-standard model, or any strong assumption, to tie the collision resistance of the hash H to the pre-hash G.

One way to be sure that we design our hash in such a way that the hash does not fundamentally rely on randomness, or some other property, of the block cipher E_k, is that there should not be an attack on the hash when E_k is substituted by the identity function. With (3), if E_k is the identity function, then $H \equiv 0$, which is a function that has no collision resistance whatsoever. If we replace $\oplus$ in (3) with the group operation, then we get $H = 2G$, which suffers from the same preimage attacks as G. Therefore the design of H from (3) fails to meet the third of our three tenets.

[3] This function is the basis for the compression function of SHA-1 and SHA-256.

[4] We do not assert the necessity of collision resistance of F_k, because an arbitrary collision in F_k can only be assume to be random hash values, for which we do not know how to find G-preimages that would lead to a collision in H.

We now endeavor to find preimage resistant enhancements of EECH that we can show to meet all or most of our three minimalist tenets.

2.3 LEECH

The first enhancement of EECH, we call *Logarithmic*, (and abbreviate LEECH), under the idea that the preimage resistance derives from the difficulty of the discrete logarithm problem.

$$H = [G]P, \tag{4}$$

where $P \in \mathbb{G}$ is some fixed group group element (in $\mathbb{G}$, or possibly some other secure additive group), and $[G]$ is integer derived from G.

A disadvantage of LEECH is that is no longer incremental. This disadvantage can be overcome by retaining a copy of G (which is incremental) in a safe place for those authorized to increment. For example, if we wish to digitally sign the contents of a disk drive, then we can assume that an adversary has access to the message signed, so storing G on the disk should not cause any security problem. For an encrypted disk drive, obviously, one should also encrypt G.

2.4 REECH

A faster enhancement, but still a non-incremental one, is:

$$H = E_k(m + 1 \| G_-) + E_k(m + 2 \| G_+), \tag{5}$$

where G_- contains the left half of the bits of G, and G_+, the right half. We call this hash function *Repeated* enhancement (REECH), because we can view H as EECH applied to G. As with LEECH, a disadvantage of REECH is that it is not incremental, but this can often be overcome with the same technique of storing the EECH pre-hash G along the with message. Clearly, REECH is faster to evaluate than LEECH, but as the message length increases, the difference in speed does not. For long messages, the relative difference is small.

2.5 PEECH

Rather than trying to speed up LEECH, as we did in REECH, we could try to preserve δ-incrementality. To do this, we could try to derive H from G using an homomorphic one-way function F, because then $H' = F(G') = F(G + D) = F(G) + F(D) = H + F(D)$, where D is the difference needed to update $G(M)$ to $G'(M)$ in EECH. One such function uses a pairing:

$$H = \langle G, K \rangle, \tag{6}$$

where K is some fixed point for which such a pairing is defined. We call this the *Pairing* enhancement (PEECH). Pairing evaluations are generally slower than scalar multiplications, so PEECH would be slower than LEECH. Generally, pairings evaluate into a field whose size is considerably larger than the field size used for the elliptic curve (or would be used for LEECH), so the hashes obtained from PEECH would be larger. Useful pairings are generally only defined for rather special types of curves (so, for example, a NIST curve cannot be used).

2.6 CHEECH

A fairly fast, but not δ-incremental, option for deriving H is to use a fast but non-cryptographic checksum[5]. The checksum $C = C(M)$ can be a simple XOR of the message blocks, or some more sophisticated error correcting code. We compute the hash as

$$H = G + E_k(m + 1 \| C(M)) \tag{7}$$

We will call this the *Checksum* enhancement (CHEECH). In essence, CHEECH is EECH applied to the message $(M, C(M))$.

Although CHEECH is not δ-incremental, it is still incremental because, if a small change is made to M, then H may be updated as quickly as the fast checksum can be recomputed. The reason that it is not δ-incremental is that it requires the full knowledge of M to make this update, which implies that the cost of the hash update depends on the length of M. For certain environments where incrementality is useful, it is realistic to assume that the entire message is available for recomputing the checksum. Generally, CHEECH will be faster than PEECH, provided the checksum and extra group addition can be computed faster than the pairing. Only for very long messages would $C(M)$ start to cost more than a pairing. The hash size for CHEECH is smaller than PEECH. Finally, CHEECH can work over any elliptic curve or other secure group, no pairing required (so, for example, a NIST curve may be used).

3 The Ideal Cipher Model

In this model, we assume that E_k is a random bijection from bit strings of a given length l to some random subset S of the group $\mathbb{G}$. Furthermore, an adversary A can only compute E_k through access to an oracle. The adversary may submit any bit string b of length l to the oracle and receive its encryption $E_k(b) \in \mathbb{G}$.

We will also allow the adversary A to compute E_k^{-1}, but again only by invoking an oracle. The adversary can submit any group element $G \in \mathbb{G}$ to the oracle, and receive either its decryption $E_k^{-1}(G) \in \{0,1\}^l$, or indication $\perp$ that $G \notin S$.

We can simulate such an oracle for the adversary. We now describe the *default* simulation. We will maintain a list $L = (L_1, \ldots, L_q)$ of pairs $L_i = (b_i, G_i)$ where $b_i \in \{0,1\}^l \cup \{\perp\}$ and $G_i \in \mathbb{G}$, which we initialize as empty, that is $q = 0$.

Suppose that the adversary A makes an E_k query to the oracle with input $b \in \{0,1\}^l$. We first check if $b = b_i$ for $1 \leqslant i \leqslant q$. If so, then we respond with G_i. We call this an *old encryption* query. If i is the least such index we say that the query has index i. If not, then increase the length of the list to $q+1$, and set $L_{q+1} = (b_{q+1}, G_{q+1}) = (b, G_{q+1})$, where we select $G_{q+1} \in \mathbb{G} \setminus \{G_1, \ldots, G_q\}$ uniformly at random. Our response is G_{q+1}. We call this a *new encryption* query, and it has index $q+1$.

Suppose A makes a E_k^{-1} query to the oracle with input $G \in \mathbb{G}$. We first check if $G = G_i$ for some $1 \leqslant i \leqslant q$. If so, we response with b_i. We call this a *old decryption* query. If i is the least such index, we say that this query has index i. If not, then with probability $(2^l - p)/(|\mathbb{G}| - q)$, where p is the number i such that $b_i \neq \perp$, we respond with b_{q+1} which is chosen uniformly at random from $\{0,1\}^l \setminus \{b_1, \ldots, b_q\}$. Otherwise we response with $b_{q+1} = \perp$. We increment the length of our list, defining $L_{q+1} = (b_{q+1}, G_{q+1}) = (b_{q+1}, G)$. We call this a *new decryption* query with index $q+1$, and it is *valid* if $b_{q+1} \neq \perp$, otherwise it is *invalid*.

[5]This idea is due to René Struik

More generally, for any interaction of an algorithm with an ideal cipher oracle, whether or not the default simulation above, we can define old and new queries, similarly, as well indices of queries.

4 Collision Resistance

We first adapt the proof [BM97] for the collision resistance of MuHASH to EECH, and its preimage-resistant variants. In adapting the proof of MuHASH, we do not account for decryption queries.

Theorem 4.1. *Let E_k be an ideal cipher. Let A be an algorithm that finds, with probability p, a collision in EECH using q encryption queries (but no decryption queries) to the ideal cipher oracle. We give a construction of an algorithm B (in the proof) that finds discrete logarithms in the group $\mathbb{G}$. The probability of success for B is at least about $p - n^{-1}$. The cost of B is (excluding the cost of A) is at most about $2Q$ scalar multiplication in the group $\mathbb{G}$, where $Q \leqslant q + m + m'$, and m and m' are the block lengths of the encodings of the messages M and M' produced by A.*

Proof. Consider an instance (X, Y) of the discrete logarithm problem where the goal is to find z such that $Y = zX$. Algorithm B will use A to find z. Algorithm A expects to use an oracle E_k for the ideal cipher. Algorithm B will provide a simulation of such an oracle, as follows.

To generate the new encryption query response G_i, algorithm B selects x_i and y_i uniformly at random and computes

$$G_i = x_i X + y_i Y \tag{8}$$

repeating as necessary to ensure that G_i is distinct from previous group elements in its list. In all other respects, the simulated oracle acts as the default simulation does. Because G_i has a uniform distribution, this simulation is actually indistinguishable from the default simulation. So, by its definition, with probability p, A finds a collision, which is to say, distinct messages M and M' giving rise to the same hash value H. Let m and m' denote the number of blocks in the encodings of M and M', respectively.

To verify the alleged collision pair M and M', algorithm B continues its simulation of the oracle to check that $G = G'$. This entails making at most m new encryption queries to find pre-hash G, and m' for G'.

$$G = \sum_{j=1}^{m} E_k(j \| M_j), \qquad G' = \sum_{j=1}^{m'} E_k(j \| M'_j)$$

Let Q the total number of new queries made. Then $Q \leqslant q + m + m'$. Algorithm B makes the encryption queries for M first, in ascending order of index j, and then queries for M', with j in ascending order.

Let $i(j)$ be the index of encryption query $E_k(j \| M_j)$ and let $i'(j)$ be the index of query $E_k(j \| M'_j)$. Then we have

$$G = \sum_{j=1}^{m} x_{i(j)} X + y_{i(j)} Y, \qquad G' = \sum_{j=1}^{m'} x_{i'(j)} X + y_{i'(j)} Y$$

For $1 \leqslant i \leqslant Q$, let $j(i)$ be the number j such that $i(j) = i$. Define $j'(i)$ similarly. Then:

$$G = \sum_{i=1}^{Q} j(i) x_i X + j(i) y_i Y, \qquad G' = \sum_{i=1}^{Q} j'(i) x_i X + j'(i) y_i Y$$

If $G = G'$, then computing $G - G'$ and separating the X and Y contributions, we conclude that

$$-\sum_{i=1}^{Q}(j(i) - j'(i))x_i \equiv \left(\sum_{i=1}^{Q}(j(i) - j'(i))y_i\right) z \bmod n$$

Here n is the order of the group $\mathbb{G}$, which we assume is prime. From this equation, we can solve for z, unless the factor in front of z is zero (modulo n).

We now claim that algorithm A has no information about the y_i, and in particular, the y_i are independent of $j(i) - j'(i)$. Of course, algorithm B chose y_i independently of earlier choices of A, but $j(i) - j'(i)$ can depend on what A does after seeing $G_i = x_i X + y_i Y$. However, G_i does not reveal anything about y_i because for each possible value of y_i there is a possible value of x_i. Given this claim that the y_i are independent of the $j(i) - j'(i)$, the probability that

$$\sum_{i=1}^{Q}(j(i) - j'(i))y_i \equiv 0 \bmod n$$

is at most $1/n$, provided that for at least one value of i, we have $j(i) - j'(i) \not\equiv 0 \bmod n$. In this case B fails, and A has been extremely lucky.

Otherwise, $j(i) \equiv j'(i) \bmod n$ for all i. Note that $0 \leqslant j(i), j'(i) \leqslant 1$ because the counters j are increasing. Therefore we have $j(i) = j'(i)$ for all i, which implies that $M = M'$. □

Note that the proof above is really the conjunction of two proofs from [BM97]: a first proof that MuHASH is secure in the random oracle model if the *balance problem* is hard, and a second proof that the balance problem is hard if the discrete logarithm problem is hard. The balance problem is: given a random set of group elements, find two distinct subsets whose sum is the same. In the same spirit we have:

Theorem 4.2. *Function EECH is collision-resistant in the sense of Theorem 4.1 if the balance problem for $\mathbb{G}$ is hard.*

Proof. The proof is similar to Theorem 4.1 and the proof that MuHASH is secure under balance problem from [BM97]. □

Hardness of the balance problem is much closer to a necessary condition for the collision resistance of both EECH and MuHASH than the discrete logarithm is. If the balance problem is solvable in $\mathbb{G}$, then in the ideal cipher model or random oracle model, the balance problem solver can be used to find a collision in EECH or MuHASH, respectively.[6]

Hardness of the discrete logarithm problem, however, does not appear to be necessary for hardness of the balance problem, or for the collision resistance of MuHASH, as discussed in [BM97], nor does it appear necessary for EECH.

4.1 Collision Resistance With Decryption Queries

Now we deal with the possibility that the adversary makes decryption queries, which, of course, is not at all unrealistic, since k is fixed and known. Consider the following collision finding algorithm

[6] Also, if k is selected at random, and we can solve the balance problem, then either we can find a collision in EECH or the block cipher E_k is insecure. A similar result holds for MuHASH. That said, for fixed k, it is possible that a balance problem solver is not useful for finding a collision in EECH or MuHASH.

A that uses just a *single* decryption query. Choose any message M, where $m \geqslant 3$ and compute its EECH hash (pre-hash) G. Now A finds some colliding message M' as follows. Let $M'_j = M_j$ for $j \geqslant 3$. Choose $M'_1 \neq M_1$ at random and choose M'_2 by

$$j' \| M'_2 = E_k^{-1}\left(E_k(1\|M_1) + E_k(2\|M_2) - E_k(1\|M'_1)\right) \tag{9}$$

If $j' = 2$, then M' will form a collision with M. In the ideal cipher model, we can assume that j' is a random c-bit integer where c is some fixed value. The probability that $j' = 2$ is then 2^{-c}. The adversary can also repeat this process, trying on the about 2^c random values of M'_1 to guarantee that one giving $j' = 2$ will almost certainly be found. To prevent such an attack, we therefore need to choose c to make such an attack infeasible. Choosing $c \approx l/2$, seems to be an effective way to do this, because 2^c iterations will be too much for the adversary (if $n \approx 2^l$). Incidentally, with $c \approx l/2$, there is a good chance that there is no $M'_1 \neq M_1$ gives $j' = 2$.

In other words, any proof of collision resistance against an adversary who makes decryption queries must depend on the plaintexts containing sufficient redundancy. Intuitively, the redundancy eliminates the any advantage the adversary gets from a decryption query. Of course, the redundancy requires dividing the message into more blocks, and thus requires more elliptic curve additions.

Theorem 4.3. *Suppose that adversary A finds a collision in EECH, when implemented with an ideal cipher E_k. Suppose that EECH is implemented with c bits of redundancy per plaintext. Assume that A has probability p of succeeding, and that A makes q queries to the ideal cipher oracle. Then A can be used to build an algorithm B that solves the discrete logarithm problem in $\mathbb{G}$. The probability of B succeeding is at most $p - n^{-1} - (1 - (1 - 2^{-c})^q)$, and the cost of B is about $2Q$ scalar multiplication in $\mathbb{G}$, where $Q \leqslant q + m + m'$, where m and m' are the block lengths of the colliding messages found by A.*

Proof. We use the same construction for B as in Theorem 4.1. We will look at the event that every decryption query made by A yields an output that is invalid, in the sense that it cannot be used to form a plaintext block corresponding to a message. In this event, the decryption queries give A no advantage, so the proof of Theorem 4.1 applies. Each decryption query has probability of 2^{-c} of being valid. The probability that at least one of q distinct decryption queries is valid is $1 - (1 - 2^{-c})^q$. □

To apply this theorem, note that if $q \approx 2^c$, then $(1 - 2^{-c})^q \approx e^{-1}$, where e is the base of the natural logarithm. Therefore if A has probability p of success significantly larger than $1 - e^{-1}$, then B will have significant probability of success. An attacker could try 2^c encryption queries and 2^c decryption queries of the form prescribed in (9). With probability about $1 - (1 - 2^{-c})^{2^c} \approx 1 - e^{-1}$, one of them of will yield a collision. Therefore, the bound in Theorem 4.3 is close to optimal.

4.2 Collision Resistance of prEECH

We need to show that our various prEECH constructions do not hinder the collision resistance of EECH.

Theorem 4.4. *LEECH, REECH, PEECH, and CHEECH are each as collision resistant as EECH, in the sense that a collision in a prEECH leads to a collision in EECH.*

Proof. We consider three arguments:

1. Functions LEECH and PEECH are the composition of an injection I with G (EECH). If (M, M') is a collision in LEECH or PEECH, meaning $I(G(M)) = I(G(M'))$, then $G(M) = G(M')$, because I is injective. Therefore (M, M') is a collision in EECH.

2. Function REECH is essentially EECH applied twice. If (M, M') is a collision in REECH, meaning $G(G(M)) = G(G(M'))$, then (M, M') or $(G(M), G(M'))$ is a collision in EECH.

3. Function CHEECH is essentially EECH applied to the message with a checksum added. If (M, M') is a collision in CHEECH, meaning $G(M, C(M)) = G(M', C(M'))$, then $(M, C(M))$ and $(M', C(M'))$ is a collision in EECH.

In all three cases, we have an immediate deduction to a collision in EECH. □

5 Preimage Resistance

A folkloric argument states that a collision resistant hash function is also preimage resistant: If not, just pick some random message, compute its hash, and then find a preimage, which gives a collision with the original message. This argument is contingent, however, on some implicit assumptions about the hash function. For example, the argument fails if the hash is only preimage resistant on the hashes of random messages, but has a pathological range of hash values for which a single preimage is easy to find. The Handbook of Applied Cryptography, Note 9.20, gives an example of such a hash.

One could argue, though, that most hash function designs would not be so pathological, and therefore that the grounds above for insisting on preimage resistance (over and above collision resistance) are only theoretical. The case of EECH, we submit, is a non-pathological example of hash that may be collision resistant but is definitely not preimage resistant. Recall that if an adversary knows all of a message except for a single block, then the adversary can use G to find the missing block. In particular, if the message has only a single block, then an adversary can use G to find the whole message, provided that G is actually the pre-hash of a single block message,[7] which actually only true for a negligible proportion of random $G \in \mathbb{G}$. Single block inversion cannot be converted into a collision attack using the folkloric argument above, because, when varying only a single message block, EECH is essentially injective, and therefore collision-free (not just collision-resistant).

The fact that EECH is the foundation for LEECH, REECH, PEECH and CHEECH — despite EECH lacking comprehensive preimage resistance — constitutes more than a theoretical ground for insisting on preimage resistance. Accordingly, we shall prove the preimage resistance of LEECH, REECH, PEECH and CHEECH. We shall also prove a form of partial preimage resistance for EECH, where the part of message unknown to the adversary is distributed over two or more message blocks.

5.1 Bijective Enhancements

We deal first the enhancements LEECH and PEECH which, recall, are obtained by applying a one-way bijection to EECH.

Theorem 5.1. *Suppose that algorithm A finds preimages of LEECH for random elements in its range with probability p. Then we can construct an algorithm B (construction given in the proof),*

[7] Single block excludes the length-encoding block.

using A as a subroutine, such that B solves discrete logarithms to be the base P (the same based used in LEECH) with total probability p.

Proof. Algorithm B is given random group element Q and needs to find u such that $Q = uP$. Algorithm A call algorithm A on input Q, which finds, with probability p a message M such that $H(M) = Q$. Algorithm sets $u = [G(M)]$ where $G(M)$ is the EECH pre-hash of M. □

Theorem 5.2. *Suppose that algorithm A finds preimages of PEECH for random elements in its range with probability p. Then we can construct an algorithm B (construction given in the proof), using A as a subroutine, such that B inverts the pairing function $\langle \cdot, K \rangle$ with total probability p.*

Proof. Algorithm B is given V and needs to solve for U such that $V = \langle U, K \rangle$. Algorithm B calls algorithm A on input $H = V$, using the default simulation of the encryption oracle. When A outputs an alleged preimage M, algorithm B continues running the encryption oracle to compute the pre-hash G. Finally, B outputs $U = G$. □

Note that if the pairing is a *symmetric pairing* on the group $\mathbb{G}$, then inverting the pairing is as difficult as the (computational) Diffie-Hellman problem, since, if we put $V = \langle xK, yK \rangle$, then we have $U = xyK$, in the notation of the proof above.

5.2 Partial Preimage Resistance of EECH Itself

While EECH does not have full preimage resistance, we can nevertheless prove that EECH does have a partial form of preimage resistance. First, some lemmas.

Lemma 5.1. *Suppose that a function $H : X \to Y$ has the property that for random $x \in X$, then $\#H^{-1}(H(x)) \geqslant 2$, with probability q. Let A be an algorithm such that for random $x \in X$, given $H(x)$ algorithm A finds x' such that $H(x') = H(x)$ with probability p. Suppose that the cost of A it t. Then an algorithm B can be built using A that finds a collision (x, x') in H, meaning $x \neq x'$ but $H(x) = H(x')$, with probability at least $\frac{p+q-1}{2}$ and cost $t + \epsilon$, where ϵ is the cost of sampling $x \in X$ plus computing H.*

Proof. Algorithm B works as follows. First, it selects random $x \in X$. Second, it computes $h = H(x)$. Third, it calls A on input h, which outputs x'. Finally, B outputs (x, x').

By definition of A, with probability p we have $H(x') = h$. We will assume this event. By our assumption on H, with probability at most $1 - q$, we have $\#H^{-1}(h) = 1$. If we exclude this event, then we get a probability of at least $p + q - 1$, for the event that $H(x') = h$ and $\#H^{-1}(h) \geqslant 2$. In this event, the probability that $x = x'$ is at most $\frac{1}{2}$. Therefore with probability at least $\frac{p+q-1}{2}$, we will have the desired collision. □

In essence, this simple lemma is the basic folkloric argument, with a non-degeneracy criterion for the hash function. (A form of this argument has been described by Stinson.)

Lemma 5.2. *With the same assumptions as Lemma 5.1, suppose that $\#X = r\#Y$, then $q \geqslant 1 - r^{-1}$.*

Proof. At most $\#Y$ elements x of X are such that $\#H^{-1}(x) = 1$. □

We now suppose that in EECH the number of values of each message block is about $\sqrt{n}$, so that half the plaintext bits are used for the message, the other half for the index and counters.

Theorem 5.3. *Let A be an algorithm that does the following. Algorithm A is given $G(M)$ where at least three message blocks have been selected uniformly at random. Algorithm A then finds message M' such that $G(M') = G(M)$, with probability p and cost t. Then A can be used to construct an algorithm B that finds a collision in G with probability $p' \approx p/2$ and cost $t' \approx t$.*

Proof. Suppose that blocks M_i, M_j, M_l are selected at random. Let X be the space of all such triples (M_i, M_j, M_l). Let $Y = \mathbb{G}$. Fixing all other message blocks, EECH defines a function $H : X \to Y$, as in Lemma 5.2 with $r \approx \sqrt{n}$. Applying Lemma 5.1, we have $p' \approx \frac{p+1-1/\sqrt{n}-1}{2} \approx p/2$.

If more than three blocks are randomized, apply the result for above for each fixed value of the other randomized blocks, or just use a larger value of r. □

Of course, it should be noted that one generally expects a higher security level against preimage attacks than against collision attacks. This result does not provide such an increase in security level.

If only two blocks are randomized, then we do not have such a simple counting argument, because we get $r \approx 1$. If more the plaintext blocks are devoted to the message, then we do get $r > 1$, but then we risk an easier collision attack. Instead, we can use the ideal cipher model.

Theorem 5.4. *Let A be an algorithm that does the following. It is given $G(M)$ where exactly two blocks of M have been selected uniformly at random. Suppose A finds a message M' such that $G(M') = G(M)$ with probability p and cost t, where G is EECH instantiated in the ideal cipher model. Then A be used to devise an algorithm B that finds a collision in EECH with probability ... and cost ...*

Proof. Algorithm B provides a simulation of the ideal cipher oracle for A, which is then continued to verify that $G(M) = G(M')$. Of course (M', M) is a collision, unless $M' = M$. Let i and j be indices of the message blocks of M chosen at random. We need to show that $G(M)$ reveals nothing (M_i, M_j) and therefore that A has negligible probability of producing $M' = M$.

$$G(M) = E_k(i\|M_i) + E_k(j\|M_j) + H, \tag{10}$$

where H is evaluated over a fixed remainder of the message. Algorithm choose a random point $G \in \mathbb{G}$ and gives this to A as the value of $G(M)$. For the simulation of the ideal cipher oracle, algorithm B uses the default simulation with exception of the second of new encryption queries $(i\|M_i)$ and $(j\|M_j)$. Suppose that the first response to one of this is a random $R \in \mathbb{G}$. The second response is $G - R - H$.

If A does not make encryption queries to at least one of $(i\|M_i)$ and $(j\|M_j)$, or a decryption query to R or $G - R - H$, then M_i and M_j can take any value, they can be decided by B after A is done, during the verification of $G(M') = G(M)$. On the other hand A has negligible chance of guessing one of the four fixed values above. □

Corollary 5.1. *Let A be an algorithm that on input of a uniformly random $G \in \mathbb{G}$, finds an EECH pre-image of G, when instantiated in the ideal cipher model. That is, a message M such that $G(M) = G$. Then A can be used to construct an algorithm B that finds a collision in EECH.*

Proof. The hashes of single block messages form a negligible small portion of $\mathbb{G}$, so for random $G \in \mathbb{G}$, any preimage M has with overwhelming probability at least two blocks. Now the previous result may be applied. □

5.3 Non-Bijective Enhancements of EECH

Corollary 5.2. *CHEECH is preimage resistant, even if only a single block in unknown to the adversary.*

Proof. Suppose that an adversary A can find preimages in CHEECH, with only one unknown single block. Then $(M, C(M))$ has two blocks that are unknown to the adversary, by the properties of $C(M)$ (such as being the sum of the all blocks in some efficiently computable group). In essence, the CHEECH of M is the EECH of $(M, C(M))$. Inverting CHEECH on M with one secret block, is essentially inverting EECH on $(M, C(M))$ with two secret blocks and extra condition the the checksum must be consistent. But we already saw that inverting EECH with two secret blocks is infeasible. □

Theorem 5.5. *REECH is preimage resistant,*

Proof. Suppose that an adversary A finds a REECH preimage M of random $H \in \mathbb{G}$. Then $G(M)$ is a two-block EECH preimage of random $H \in \mathbb{G}$. This is infeasible. □

6 Security of Cipherless Variants in the Generic Group Model

The ideal cipher model enabled us to find security proof, but it was our objective that the security should not require the block cipher to be secure. There seems to be no good fundamental reason that a secure block cipher is needed to construct a secure hash function. Indeed, there seems to be no good fundamental reason that a block cipher (with a non-secret key) could help security.[8]

Conjecture 6.1. *If the block cipher in EECH is replaced by the identity function, the result is collision resistant. With the same replacement, the prEECH functions remain collision and preimage resistant.*

We call EECH with E_k instantiated by the identity function the Elliptic Curve Only Hash (ECOH). We define Logarithmic, Repeated, Pairing and Checksum ECOH, analogously.

Our proofs that depend on the ideal cipher model proofs are not really applicable to ECOH and prECOH, because the identity function seems quite inappropriate to model as a random. Some of our proofs do not use the ideal cipher model, so these still apply. In particular, Theorems 4.4 and 5.1 still apply.

Another non-standard model, the generic group model can be used to prove the collision resistance of ECOH. This should not be too surprising, given a generic group has randomized representations, which is not unlike encrypting the representation.

Theorem 6.1. *In the generic group model, EECH is collision resistant, provided that E_k is bijective. In particular, ECOH is collision resistant in the generic group model.*

Proof. Suppose that algorithm A finds a collision in EECH when the group is implemented as a generic group. Suppose that M and M' are the colliding messages that A finds. Provided that A does not make too many queries, then the none of the outputs of the generic group oracle (where the input queries consist of previous outputs of the generic group oracle) will have a representation consistent with the embedding of message blocks into the group.

[8] Well, actually, hash functions are keyless and are often perceived as contributing to security, so it only seems fair to think of keyless permutations as contributing to security, on a similar basis.

A realistic version of the generic group model for EECH must allow oracle to accept arbitrary bit string representations of group elements, and these must have reasonable chance of being valid (else ECOH would not be feasible). To verify the collision of M and M', one invokes the generic group oracle. One finds that all of the message blocks, represented as group elements, are new inputs, and thus are assigned random discrete logarithms by the generic group oracle. The sums of two sets of random numbers have negligible chance of being equal. □

Acknowledgments

I thank Scott Vanstone and René Struik for insightful comments. René Struik, on seeing the design of PEECH, proposed the design of CHEECH.[9] I thank Dan Bernstein for commenting about issues about the efficiency of embedding bit strings into an elliptic curve group.

References

[BM97] Mihir Bellare and Daniele Micciancio, *A new paradigm for collision-free hashing: Incrementality at reduced cost*, Advances in Cryptology — EUROCRYPT '97 (Walter Fumy, ed.), Lecture Notes in Computer Science, no. 1233, Springer, 1997, Full version: `http://eprint.iacr.org/1997/001`, pp. 163–192.

A TEECH

Each of the functions above has outputs in some cryptographic group. As such, the hash value may have properties making it distinguishable from a random bit string. We now seek to provide a hash function derived from EECH whose output appears more random:

$$H = T_t(E_k(G)), \tag{11}$$

where T is a function that truncates t bits from a bit string. We call this enhancement Truncated EECH (TEECH).[10] The greater t, the better the pseudorandomness, but the larger the group $\mathbb{G}$ that must be used, which makes its computation slower than the others. Another possible disadvantage is that TEECH is not incremental, unless we keep a copy of the pre-hash G together with the message (as we proposed to do for LEECH and REECH).

Function TEECH, is a little different in design from the LEECH, PEECH, REECH and CHEECH, because there is significant information loss after the EECH. If t is set too high, then one can find collisions in TEECH by the birthday paradox, and preimages are also easier to find. So, TEECH fails to meet our second tenet of minimalism: there is no direct translation from a collision in TEECH to a collision EECH. However, in the ideal cipher model, we can formulate such a translation.

Theorem A.1. *Provided that t is not too low, and algorithm A finds a collision in TEECH by working in the ideal cipher model (only respect to the conversion of EECH to TEECH), then an algorithm B may be constructed that finds a collision in EECH.*

[9] Though, René is not to be blamed for the moniker.

[10] The idea of truncating MuHASH was discussed in [BM97].

Proof. Algorithm B will simulate for A an ideal cipher oracle as follows. For encryption oracle query with input $G \in \mathbb{G}$, algorithm B outputs a random bit string of length l (the block size of the cipher). For decryption oracle query with input $b \in \{0,1\}^l$, output a random $G \in \mathbb{G}$. This is essentially the default simulation mentioned earlier, but note that in the opposite direction to the one used for EECH because EECH uses the cipher in the opposite direction to TEECH (bit strings to group elements). By its definition, A will find a collision, say (M, M').

Algorithm B now verifies the collision $T_t(E_k(G(M))) = T_t(E_k(G(M')))$, where G is the function EECH. The verification is done by continuing its simulation of the ideal cipher oracle. If:

1. $G(M)$ and $G(M')$ both first appeared in the oracle interactions as inputs to new encryption queries (either made by A or by B in the verification of the collision),
2. $G(M) \neq G(M')$, and
3. t is not too small,

then the probability that of being a collision is negligible, because here B will selected $E_k(M)$ and $E_k(M')$ as random bit strings of length l, making the their probability of colliding 2^{l-t}. If $G(M) = G(M')$, we have found our collision in EECH, and by the hypothesis t is not too small.

The only remaining case is that at least one of $G(M)$ or $G(M')$ did not first appear as a new input query, say $G(M)$. Then $G(M)$ first appears as the output of a decryption query, with input say b. Now $G = G(M)$ was selected as a random point, and algorithm A somehow found an EECH preimage M to G. In other words, the only remaining case is that A can actually invert EECH on random values in $\mathbb{G}$. We will prove later that this is infeasible. □

Another important goal of TEECH is preimage resistance. To prove this, we may again be able to resort to the ideal cipher model.

We note that from our minimalist perspective, TEECH is preferable to the function $G + E_k(G)$, where G is EECH, because by making the block cipher an identity function, TEECH appears to be preimage resistant, whereas $G + E_k(G) = 2G$ is definitely not.