Multicollision attacks on iterated hash functions

Douglas R. Stinson

David R. Cheriton School of Computer Science
University of Waterloo

Fourth Pythagorean Conference
Corfu, Greece
Friday, June 4, 2010

references and summary

This talk is based on joint work with Mridul Nandi and Jalaj Upadhyay:

- M. Nandi and D.R. Stinson. Multicollision attacks on some generalized sequential hash functions. *IEEE Transactions on Information Theory* **53** (2007), 759–767.
- D.R. Stinson and J. Upadhyay. On the complexity of the herding attack and some related attacks on hash functions. *IACR ePrint 2010/30*.

I will talk about two recent results on multicollision attacks for hash functions:

1. a generalization of Joux's multicollision attack to a wide variety of hash functions, and
2. a second look at constructing diamond structures, which were invented by Kelsey and Kohno to use in their herding attacks on iterated hash functions.

hash functions

- Typically, a hash function takes a "long" input string and produces a random-looking "short" output string called a message digest.
- Hash functions have been used for many years in computer science to create hash tables for efficient methods for information retrieval.
- In this context, it is important that collisions occur as infrequently as possible, where a collision for a hash function $hash$ is a pair of distinct inputs x, x' such that $hash(x') = hash(x)$.
- Hash functions are also used frequently in cryptography, where additional properties are required. Such hash functions are termed cryptographic hash functions.
- A cryptographic hash function maps an arbitrary-length input string to a fixed-length output string: $hash : \{0,1\}^* \rightarrow \{0,1\}^n$.

three security properties of hash functions

Collision resistance

It should be difficult to find $x, x' \in \{0,1\}^*$ such that $x' \neq x$ and $hash(x') = hash(x)$.
(Here, x and x' collide.)

Preimage resistance

Given $z \in \{0,1\}^n$, it should be difficult to find $x \in \{0,1\}^*$ such that $hash(x) = z$.
(Here, x is a preimage of z.)

Second preimage resistance

Given $x \in \{0,1\}^*$, it should be difficult to find $x' \in \{0,1\}^*$ such that $x' \neq x$ and $hash(x') = hash(x)$.
(Here, x' is a second preimage of $h(x)$.)

difficulty of the three problems

- Suppose we postulate the existence of an "ideal" hash function that outputs a random value $hash(x)$ for every input x.
- Such a hash function is called a random oracle.
- It is easy to analyse the difficulty of the three problems in the random oracle model.
- Preimages and Second preimages can be found by exhaustive search in expected time $\Theta(2^n)$.
- Collisions can be found using the birthday paradox in expected time $\Theta(2^{n/2})$.
- When we construct a "real" hash function, our goal is that the three problems cannot be solved more quickly than in the ideal case (but proving things like this are extremely difficult!).

multicollisions

- There has been recent interest in studying the difficulty of finding multicollisions in hash functions.
- A γ-multicollision is a γ-subset $\{x_1, \ldots, x_\gamma\} \subseteq \{0,1\}^*$ such that $hash(x_1) = hash(x_2) = \cdots = hash(x_\gamma)$.
- It is commonly asserted that the complexity of finding a γ-multicollision in the random oracle model is $\Theta(2^{n(\gamma-1)/\gamma})$.
- Using estimates due to Diaconis and Mosteller (1989), Nandi and Stinson observed that the true complexity is $\Theta(\gamma\, 2^{n(\gamma-1)/\gamma})$.
- For additional, more detailed analysis along these lines, see Suzuki, Tonien, Kurosawa, and Toyota (2008).

iterated hash functions

- The most common design strategy for hash functions is the iterated hash function.
- MD4, MD5, and SHA-1 are all iterated hash functions.
- We need a padding function, which takes an input string x, where $|x| \geq n + t + 1$, and constructs a "padded" string y, such that $|y| \equiv 0 \bmod t$.
- We also need a compression function, $compress : \{0,1\}^{n+t} \rightarrow \{0,1\}^n$.
- IV is a public initial value which is a bitstring of length n.

constructing an iterated hash function

preprocessing step

Given x, construct the padded string y, where $|y| \equiv 0 \bmod t$. Denote

$$y = y_1 \parallel y_2 \parallel \cdots \parallel y_r,$$

where $|y_i| = t$ for $1 \leq i \leq r$. The y_i's are called message blocks.

processing step

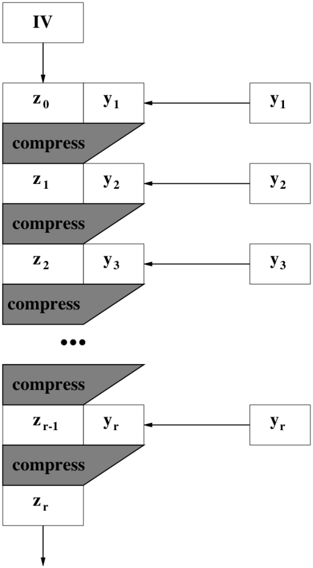
Compute the following chaining values:

$$\begin{aligned} z_0 &\leftarrow IV \\ z_1 &\leftarrow \mathit{compress}(z_0 \parallel y_1) \\ \vdots &\quad \vdots \quad \vdots \\ z_r &\leftarrow \mathit{compress}(z_{r-1} \parallel y_r). \end{aligned}$$

output

Define $h(x) = z_r$.

constructing an iterated hash function



Joux's multicollision attack

- Joux (2004) discovered a simple multicollision attack on iterated hash functions.
- The expected complexity to find a 2^r-multicollision is $\Theta(r\,2^{n/2})$, which is much smaller than the birthday attack having complexity $\Theta(2^r \times 2^{n(2^r-1)/2^r})$.
- The idea is to find r successive collisions in the compression function, each of which requires time $\Theta(2^{n/2})$ to find.
- For $z, z' \in \{0,1\}^n$ and $y \in \{0,1\}^t$, we use the notation $z \xrightarrow{y} z'$ (a labelled arc) to mean $compress(z, y) = z'$, where $|z| = |z'| = n$ and $|y| = t$.
- We can extend this notation in a natural way to incorporate multiple message blocks, e.g., $z \xrightarrow{y_1, y_2, y_3} z'$.

Joux's multicollision attack (cont.)

$$\begin{array}{llll} z_0 \xrightarrow{y_1^1} z_1 & \text{and} & z_0 \xrightarrow{y_1^2} z_1 & \text{for some } z_1\text{, where } y_1^1 \neq y_1^2 \\ z_1 \xrightarrow{y_2^1} z_2 & \text{and} & z_1 \xrightarrow{y_2^2} z_2 & \text{for some } z_2\text{, where } y_2^1 \neq y_2^2 \\ \vdots & & & \\ z_{r-1} \xrightarrow{y_r^1} z_r & \text{and} & z_{r-1} \xrightarrow{y_r^2} z_r & \text{for some } z_r\text{, where } y_r^1 \neq y_r^2. \end{array}$$

Then the set

$$\{y_1^1, y_1^2\} \times \{y_2^1, y_2^2\} \times \cdots \times \{y_r^1, y_r^2\}$$

is a 2^r-multicollision:

Question: Can Joux's attack be generalised to other types of hash functions?

generalised iterated hash functions

- hash twice uses every message block twice: $hashtwice(y) = hash(hash(IV, y), y)$ where y is the padded message.
- That is, we process the message blocks in the order $y_1, \ldots, y_r, y_1, \ldots, y_r$.
- zipper hash processes the message blocks in the order $y_1, \ldots, y_r, y_r, \ldots, y_1$.
- Let $\mathcal{S} = \{1, 2, \ldots, r\}$ denote the set of indices of the r message blocks.
- A generalised sequential hash function (GSHF) is based on a sequence $\alpha = \langle \alpha_1, \cdots, \alpha_s \rangle$ where $\alpha_i \in \mathcal{S}$ for all i.
- The GSHF based on α is defined as follows:

$$\begin{aligned} z_0 &= IV \\ z_i &= compress(z_{i-1}, y_{\alpha_i}), 1 \leq i \leq s. \end{aligned}$$

a partial order relation

- We define a relation on the symbol set $\mathcal{S}$.
- For $x, x' \in \mathcal{S}$, $x \neq x'$, define $x \prec x'$ if every occurrence of x in α precedes every occurrence of x' in α.
- The relation "$\prec$" is antisymmetric and transitive; hence "$\prec$" is a partial order.
- Two symbols $x \neq x'$ are incomparable if it is not the case that $x \prec x'$ or $x' \prec x$.
- A list of symbols $x_1, \ldots, x_d$ is a chain if $x_1 \prec x_2 \prec \cdots \prec x_d$.
- A set of chains is a chain decomposition if the chains are disjoint and their union is $\mathcal{S}$.

an attack based on a chain

- We present an attack on the hash function based on the sequence

$$\alpha = \langle 1, 2, 1, 3, 2, 4, 3, 5, 4, 5 \rangle$$

- Note that $1 \prec 3 \prec 5$ is a chain.
- We decompose α into three subsequences:

$$\langle 1, 2, 1 \rangle, \langle 3, 2, 4, 3 \rangle, \langle 5, 4, 5 \rangle$$

- Define $y_2 = y_4 = y^*$ for some arbitrary t-bit string y^*.
- The attack consists of three successive birthday attacks:

$$\begin{array}{lcl} z_0 \xrightarrow{y_1^1, y^*, y_1^1} z_1 & \text{and} & z_0 \xrightarrow{y_1^2, y^*, y_1^2} z_1 \\ z_1 \xrightarrow{y_3^1, y^*, y^*, y_3^1} z_2 & \text{and} & z_1 \xrightarrow{y_3^2, y^*, y^*, y_3^2} z_2 \\ z_2 \xrightarrow{y_5^1, y^*, y_5^1} z_3 & \text{and} & z_2 \xrightarrow{y_5^2, y^*, y_5^2} z_3 \end{array}$$

- We get a 2^3-multicollision with collision value z_3.

an attack based on an initial interval

- For hash twice, we have $\alpha = \langle 1, 2, \ldots, r, 1, 2, \ldots, r \rangle$, which does not have a chain of length longer than 1.
- We have another approach, based on the fact that the first r message blocks to be processed are all different.

(1) Use Joux's multicollision attack to find a 2^r-multicollision $\mathcal{C}$ for the first r message blocks.

(2) Let $r = uv$ for "appropriate" u and v. Divide the index interval $[r+1, 2r]$ into u equal intervals, each of size v. For $i = 1, \ldots, u$, (if possible) use a standard birthday attack to find two v-tuples from the appropriate part of $\mathcal{C}$ which collide.

(3) Provided that the u birthday attacks in step (2) all succeed, we get a multicollision set (of size 2^u) for hash twice.

combining the two attacks

We consider sequences in which every symbol occurs at most twice.

The next theorem follows from Dilworth's Theorem, which states that for any a partial order "$\prec$" on a finite set $\mathcal{S}$, the maximum number of mutually incomparable elements in $\mathcal{S}$ is equal to the minimum number of chains in any chain decomposition.

Theorem (Nandi and Stinson (2007))

Let α be a sequence of elements from symbol set $\mathcal{S} = \{1, \ldots, r\}$ such that $1 \leq \mathrm{freq}(x, \alpha) \leq 2$ for all $x \in \mathcal{S}$. Suppose that $r \geq r_1 r_2$. Then one of the following holds:

1. $\mathrm{maxchain}(\alpha) \geq r_1$, *or*
2. *there exists an initial interval $[1, w]$ such that $\alpha[1, w]$ contains at least r_2 symbols each having frequency 1.*

These attacks have subsequently been extended by Hoch and Shamir (2006) to sequences where each symbol occurs at most c times, for some fixed positive integer c.

proof sketch

- Let $\rho_1 = \mathrm{maxchain}(\alpha)$.
- If $\rho_1 \geq r_1$, we're done.
- Otherwise, when $\rho_1 < r_1$, let ρ_2 denote the maximum number of incomparable elements.
- By Dilworth's Theorem, there is a chain decomposition having ρ_2 chains.
- Each chain has length at most ρ_1, so

$$\rho_2 \geq n/\rho_1 > n/r_1 \geq r_2.$$

- Take an initial subsequence of α that contains the first occurrences of the ρ_2 incomparable elements.
- This works precisely because these elements are incomparable.

the herding attack

Kelsey and Kohno (2006) described the following hash function property, presented as a game between an attacker and a challenger:

Chosen-target-forced-prefix resistance

An attacker commits to a message digest, z, and is then challenged with a prefix, P. It should be infeasible for the attacker to be able to find a suffix S such that $hash(P \parallel S) = z$.

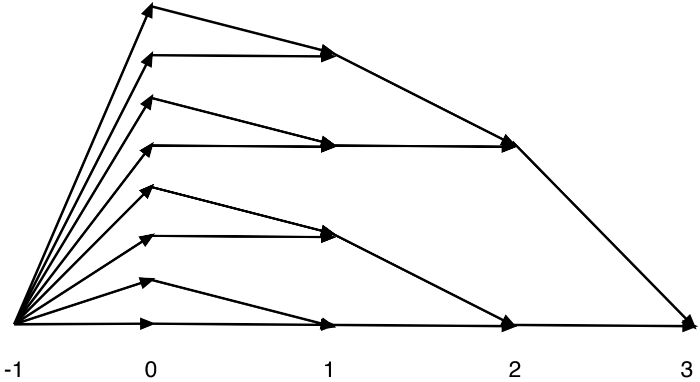
- Intuitively, it does not seem that a chosen-target-forced-prefix attack should be easier than finding a preimage, which generally takes time $\Theta(2^n)$.
- An attack that violates CTFP resistance is often called a herding attack.
- Kelsey and Kohno described a herding attack on iterated hash functions using a precomputed data structure called a diamond structure.

diamond structures

- First we'll talk about diamond structures; we'll present the herding attack a bit later.
- A 2^k-diamond structure contains a complete binary tree of depth k.
- There are $2^{k-\ell}$ nodes at level ℓ, for $k \geq \ell \geq 0$.
- There is also a single node at level -1, which we will call the source node.
- The source node is joined to every node at level 0.
- The nodes at level 0 are called the leaves of the diamond structure and the node at level k is called the root of the tree.

diamond structures (cont.)

Here is a diagram of a 2^3 diamond structure:



diamond structures (cont.)

- Every edge e in the diamond structure is labeled by a string $\sigma(e)$ which consists of one or more message blocks.
- We also assign a label $h(N)$ to every node N in the structure at level at least 0, as follows:
- Consider the unique directed path P from the source node to the node N in the diamond structure.
- P will consist of some edges $e_0 e_1 \cdots e_\ell$, where N is at level ℓ in the tree. Then we define

$$h(N) = hash(\sigma(e_0) \parallel \sigma(e_1) \parallel \cdots \parallel \sigma(e_\ell)).$$

- At any level ℓ of the structure there are $2^{k-\ell}$ hash values.
- These must be paired up in such a way that, when the next message blocks are appended, $2^{k-\ell-1}$ collisions occur.
- Thus there are $2^{k-\ell-1}$ hash values at the next level.
- The entire structure yields a 2^k-multicollision.

building a diamond structure

- A diamond structure is constructed one level at a time.
- We describe how to construct the nodes at level 1.
- For each of the 2^k nodes at level 0, construct a list of L random message blocks and compute the relevant hashes.
- Look for collisions in different lists and try to find 2^{k-1} disjoint pairs of collisions.
- For example, suppose $k = 2$, $L = 4$ and $n = 4$, and we get the following lists of hash values:

List 1:	0011	1011	0101	1100
List 2:	0010	1000	1010	0001
List 3:	0101	0001	1111	0000
List 4:	1110	1101	1011	1001

- Then we can pair up lists 1 and 4 (having collision 1011) and lists 2 and 3 (having collision 0001).

Kelsey and Kohno's analysis

Kelsey and Kohno argued as follows:

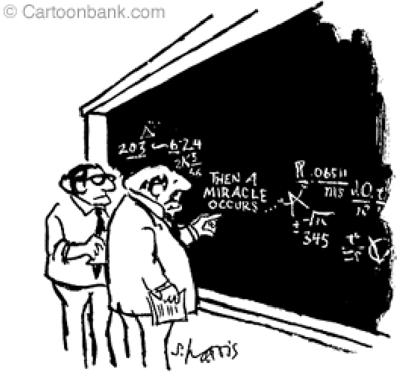
> *The work done to build the diamond structure is based on how many messages must be tried from each of 2^k starting values, before each has collided with at least one other value. Intuitively, we can make the following argument, which matches experimental data for small parameters: When we try $2^{n/2+k/2+1/2}$ messages spread out from 2^k starting hash values (lines), we get $2^{n/2+k/2+1/2-k}$ messages per line, and thus between any pair of these starting hash values, we expect about $(2^{n/2+k/2+1/2-k})^2 \times 2^{-n} = 2^{n+k+1-2k-n} = 2^{-k+1}$ collisions. We thus expect about $2^{-k+k+1} = 2$ other hash values to collide with any given starting hash value.*

the flaw in the analysis

Unfortunately, this line of reasoning does not imply that the 2^k nodes can be paired up in such a way that we get 2^{k-1} collisions:

the flaw in the analysis

Unfortunately, this line of reasoning does not imply that the 2^k nodes can be paired up in such a way that we get 2^{k-1} collisions:



"I think you should be more explicit here in step two."

random graph formulation

- It is useful to think of this problem in a graph-theoretic setting.
- Suppose we label the nodes as $1, 2, \ldots, 2^k$.
- Then we construct a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where the vertex set is $\mathcal{V} = \{v_1, \ldots, v_{2^k}\}$ and $(v_i, v_j) \in \mathcal{E}$ if the nodes v_i and v_j collide at the next level of the diamond structure.
- Let $\mathcal{G}(\nu, p)$ denote a random graph on ν labelled vertices, obtained by selecting each pair of vertices to be an edge randomly and independently with a fixed probability p.
- Based on the analysis given above, we see that the graph $\mathcal{G}$ is precisely a random graph in $\mathcal{G}(2^k, 2^{-k+1})$.
- Now, the question is if this random graph contains a perfect matching, as this is precisely what is required in order to be able to find the desired 2^{k-1} pairs of collisions.

threshold functions for random graphs

- As p increases from 0 to 1, a random graph in $\mathcal{G}(\nu, p)$ becomes more and more dense.
- Many natural monotone graph-theoretic properties become true within a very small range of values of p.
- Given a monotone graph-theoretic property, there is typically a value of p (which will be a function $t(\nu)$ depending on ν, the number of vertices) called the called threshold function.
- The given property holds in the model $\mathcal{G}(\nu, p)$ with probability close to 0 for $p < t(\nu)$, and the property holds with probability close to 1 for $p > t(\nu)$.
- A threshold function for having a perfect matching is any function having the form

$$t(\nu) = \frac{\ln \nu + f(\nu)}{\nu}$$

for any $f(\nu)$ such that $\lim_{\nu \to \infty} f(\nu) = \infty$.

fixing the analysis

- $\mathcal{G}(2^k, 2^{-k+1})$ has $p = 2/\nu$, which is much lower than required threshold, so the Kelsey-Kohno analysis is not valid.
- We assume a random graph in $\mathcal{G}(\nu, \ln \nu/\nu)$ has a perfect matching.
- We construct $\nu = 2^k$ lists, each containing L messages.
- The probability that any two given messages collide is 2^{-n}. The probability that there is at least one collision between two given lists is $p \approx L^2/2^n$.
- We want $p \approx \ln \nu/\nu$, so we take

$$L \approx \sqrt{k \ln 2} \times 2^{(n-k)/2} \approx 0.83 \times \sqrt{k} \times 2^{(n-k)/2}.$$

- The message complexity (i.e., the number of hash computations) at level 0 is therefore

$$2^k L \approx 0.83 \times \sqrt{k} \times 2^{(n+k)/2}.$$

fixing the analysis (cont.)

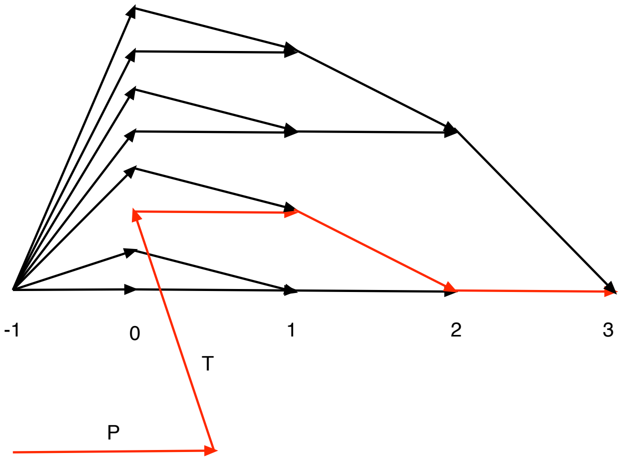
- Ignoring constant factors, this is a factor of about $\sqrt{k}$ bigger than the estimate in Kelsey-Kohno.
- The lower levels of the diamond structure are analysed in a similar way, replacing k by $k-1, k-2$, etc.
- The total message complexity is also $\Theta(\sqrt{k} \times 2^{(n+k)/2})$.
- Thus we obtain a rigourous analysis (in the random oracle model) with a precise estimate of the message complexity.
- Overall, it turns out that Kelsey and Kohno's estimate (for the entire structure) was too small by a factor of $\sqrt{k}$.
- Note this has some effect on various other attacks in the literature that make use of diamond structures.

Kelsey-Kohno's herding attack

- First, we construct a diamond structure with k levels.
- We commit to the hash value $z = h(\mathrm{root})$ and the challenger provides a prefix P.
- We choose random strings T until we find a linking message, i.e., a string T such that $hash(P \parallel T) = h(N)$ for some node N in the diamond structure.
- This takes, on average, 2^{n-k-1} attempts.
- Once we have found the linking message T, construct S by concatenating T with the message blocks in the diamond structure on the path from N to root.
- The total complexity of the attack is $\Theta(2^{n-k} + \sqrt{k} \times 2^{(n+k)/2})$.
- The value of k can be chosen as desired. If $k \approx n/3$, then the message complexity of the attack is about $\Theta(\sqrt{n} \times 2^{2n/3})$, which is a significant improvement over $\Theta(2^n)$.

Kelsey-Kohno's herding attack (cont.)

A linking message for a 2^3 diamond structure:



thank you for your attention!