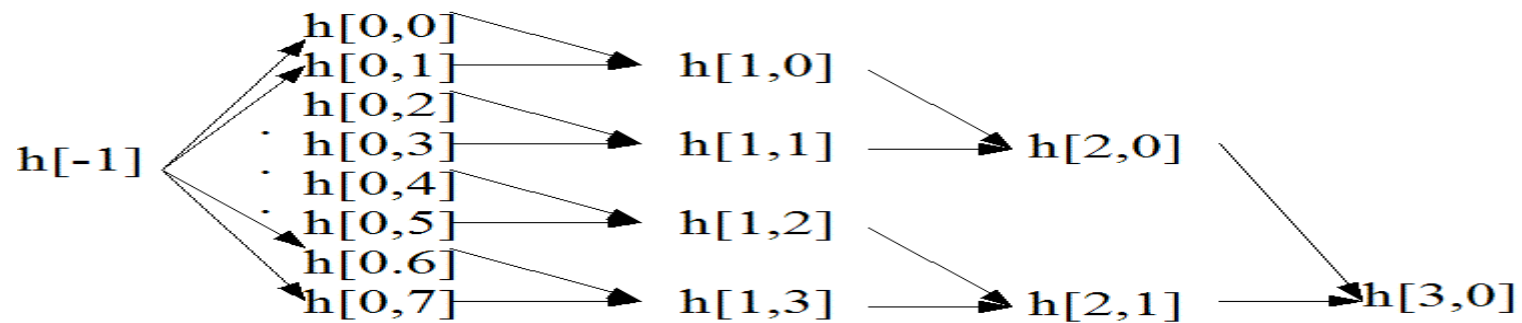# Herding Hash Functions and the Nostradamus Attack

John Kelsey, NIST

Tadayoshi Kohno, UCSD CSE Dept.

Halloween Hash Bash, 2005

A Story about Nostradamus

- In Jan 2006, Nostradamus publishes an ad in the New York Times:
  - Claims to predict many things about future, including closing stock prices for S&P500.
  - Here's the MD5 hash: *H*
- In Jan 2007, Nostradamus discloses his predictions
  - Including correct S&P 500 prices
  - Other rambling predictions
  - His predictions hash to *H*

Question: Is this evidence Nostradamus can tell the future?

Answer is NO, because of *herding*!

We can *herd* a 128-bit hash for 2^{87} work!

The Game:

1. **Precomputation:** Alice does lots of work and produces hash H.
2. Bob sends her prefix P of predefined length.
3. **Herding**: Alice produces suffix S s.t.
$$Hash(P||S) = H$$

Applies to DM-construction hash, not random oracle

For 128-bit hash:

- Precomputation costs 2^{86}
- Herding costs 2^{86}
- Suffix is 42 blocks = 2688 bytes.

What does this mean?

I can commit to hash before I know full string I'm hashing!

- With a collideable hash…
  - Proof of knowledge doesn't work
  - Collisions could involve information not available when original message signed
  - Many other surprising things can be done with collisions
- Even better results when prefix constrained!

Moral of the Story: Hash Collisions are Bad News

How to Herd a Hash Function
(in one easy lesson)

How Is It Done?

1. **Precomputation** (brute force)

   - Build a binary-tree-like search structure

     - 2^k hashes at widest point

   - Any message with one of those intermediate hashes can be herded to hash H.
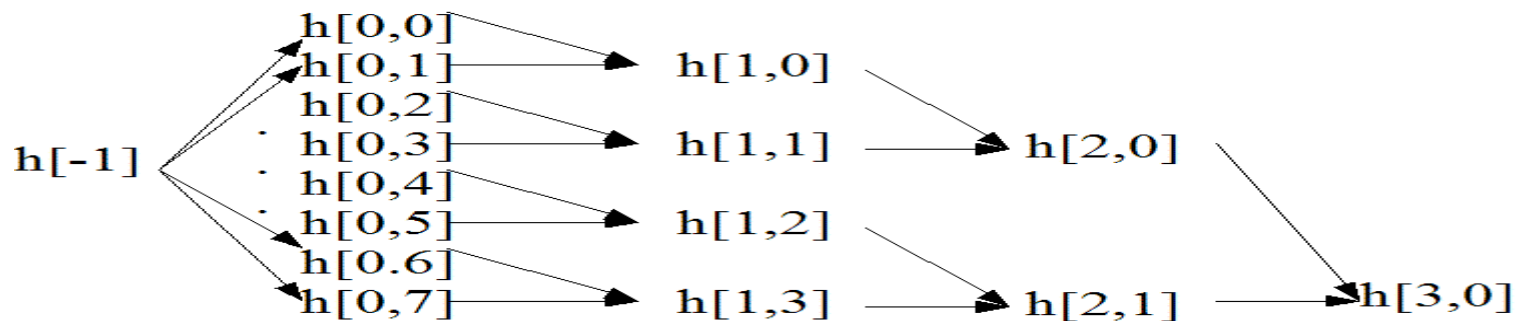
2. **Herding**

   - Brute-force search for message block to reach any intermediate hash in diamond: 2^{n-k} work.

   - Follow tree to produce rest of suffix S

To minimize work: set $W_{precomp} = W_{herd}$, solve for k=width of diamond!

Precomputation:
# Building the Diamond Structure

- Start with 2^k hash values.
  - Generate many message blocks from each
  - Find collision btwn 1st & 2nd, 3rd & 4th, etc.
- Do again with those results till we have one final hash.
  - A Merkle tree built by brute force

  Note: Arrows are message blocks, h[i,j] is hash value

```
              h[0,0]
              h[0,1] ─────►  h[1,0]
              h[0,2]                  ╲
              h[0,3] ─────►  h[1,1] ─────► h[2,0]
  h[-1]       h[0,4]                              ╲
              h[0,5] ─────►  h[1,2]                  ╲
              h[0,6]                  ╲                ╲
              h[0,7] ─────►  h[1,3] ─────► h[2,1] ─────► h[3,0]
```

Precomputation: How Much Work?

- Speedup: Don't decide which pairs of values are to collide until you find collisions

- Diamond structure with width 2^k costs about $2^{k/2+n/2+3/2}$ work.

- Diamond structure has total of $2^{k+1}-2$ intermediate hash values.

    – *Usually use only 2^k in widest level!*
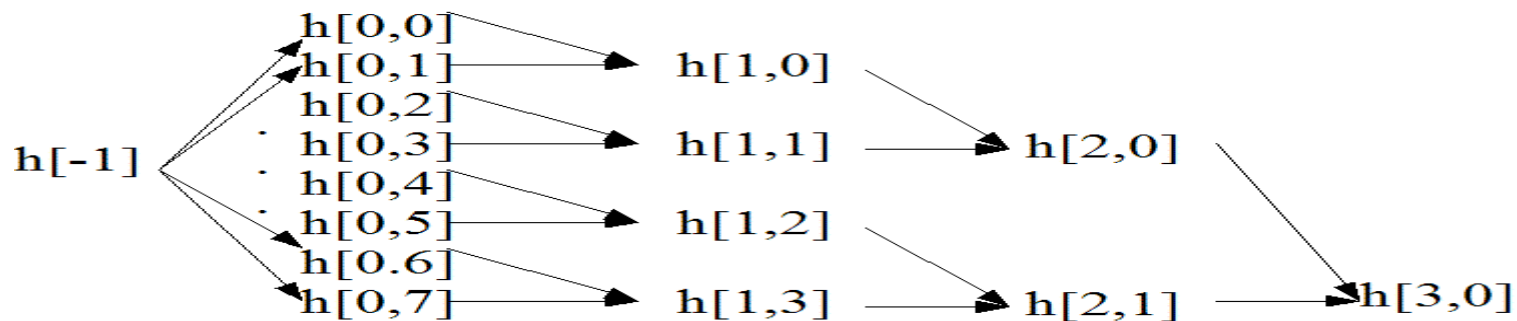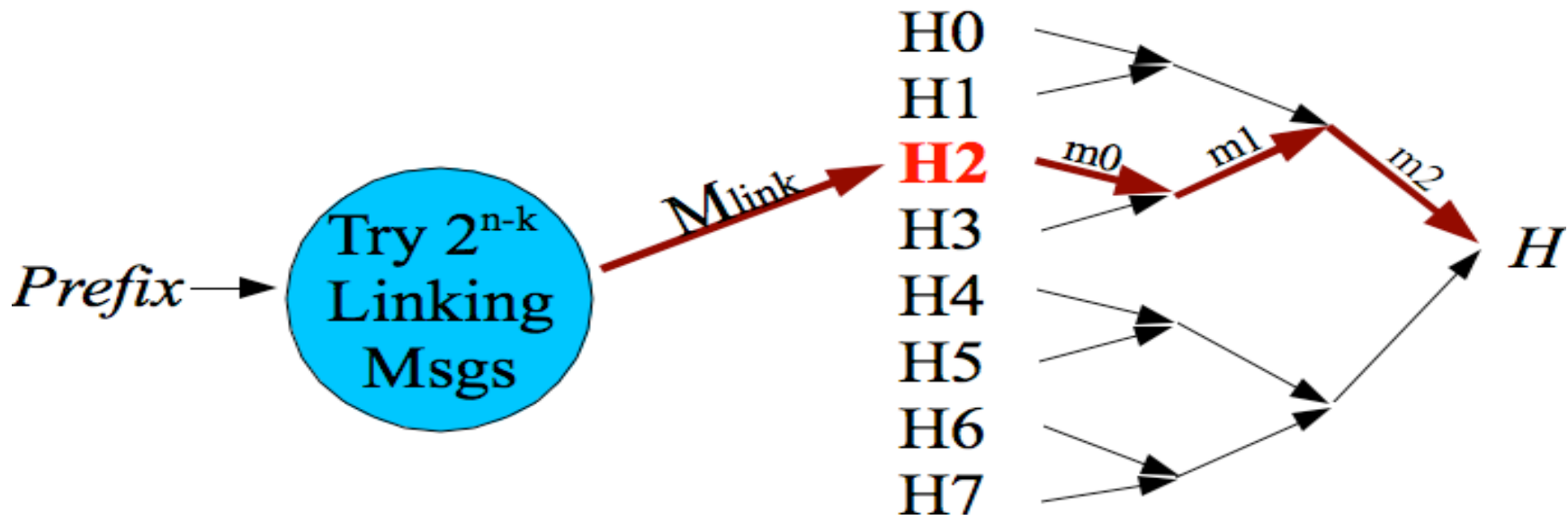
Herding a Prefix to a Hash: Finding a Match

- We have 2^k hashes in diamond structure.
- Try 2^{n-k} next message blocks
  - Match with any of 2^k hashes in widest pt.
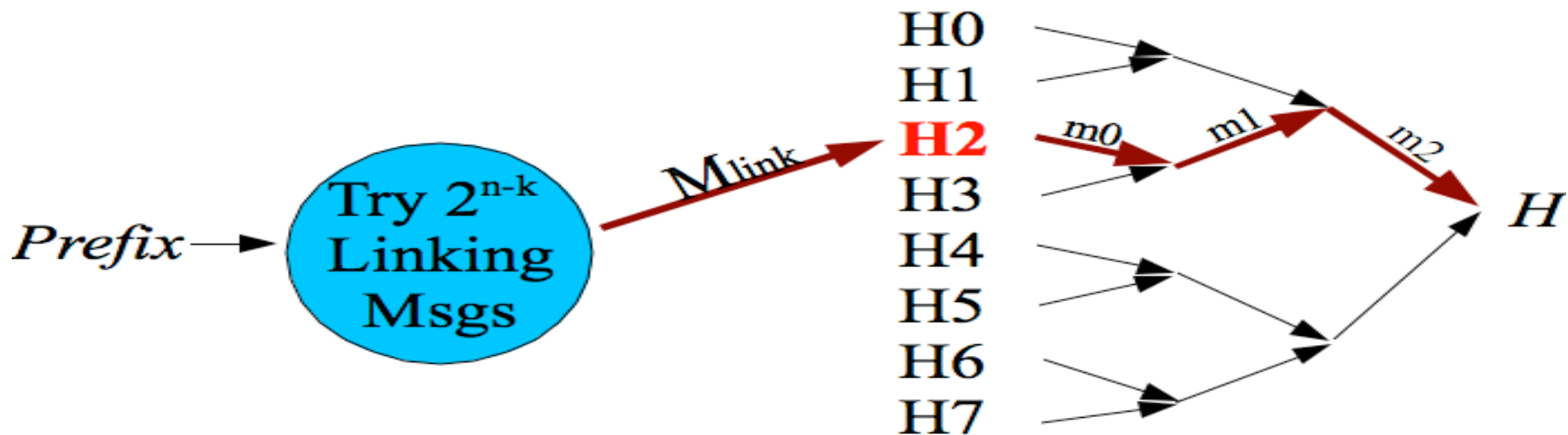
Herding: Producing Suffix

- First block of suffix is M_{link}
- Other blocks determined by diamond structure

Diagram: *Hash(Prefix || M_{link} || m_0 || m_1 || m_2)=H*

Generalizing Results

Hash Size	Lg(Tree Size) = k	Suffix Length	Work
160	52	59	2^{108}
160	15	2^{55}	2^{91}
n	(n-5)/3	k+lg(k)+1	2^{n-k}
n	(n-2r-3)/3	About 2^r	2^{n-k-r}

Note: Variants can be much cheaper!

Using Herding to Build Attacks

# Will the Suffixes be Random Bits?
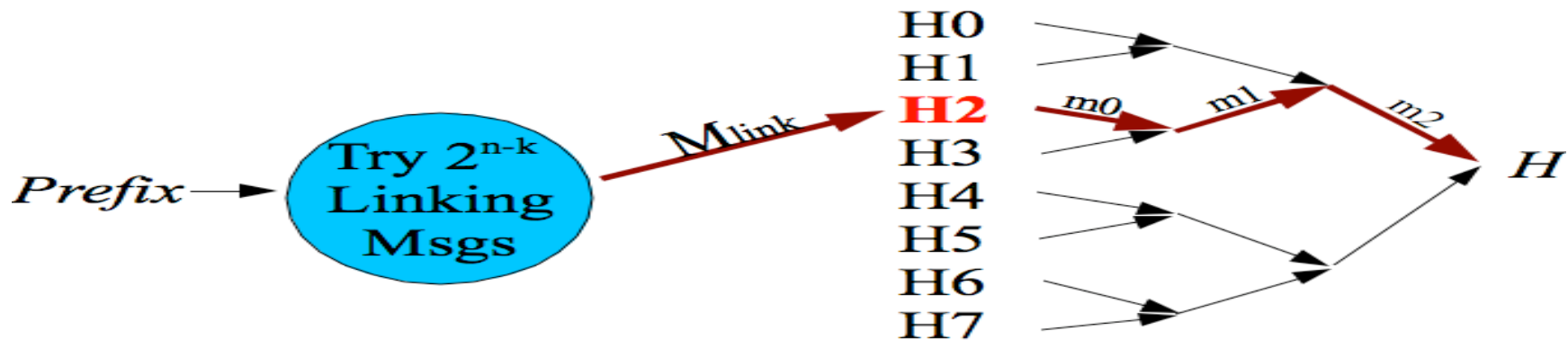
- We use Gideon Yuval's trick to make the suffix messages meaningful!

  – Generate original text, n/2 variation points, vary all $2^{n/2}$.

  – Probably requires slightly longer suffixes, perhaps 2-3 times size

  – Result: suffixes look like normal English ASCII text

Using the Basic Attack

- Pretend to know future information
 - Nostradamus Attack
  - Impersonate someone with inside knowledge
  - Attack the Guy Fawkes Protocol
- Steal inventions
  - Get timestamps on diamond structure hash.
  - To steal an invention, herd hash of its description back to your diamond structure to "prove" prior art.
- Rewrite history (despite hashes/signatures)
  - Collisions contain information not available when they were formed

Variation: Precomputing the Prefix
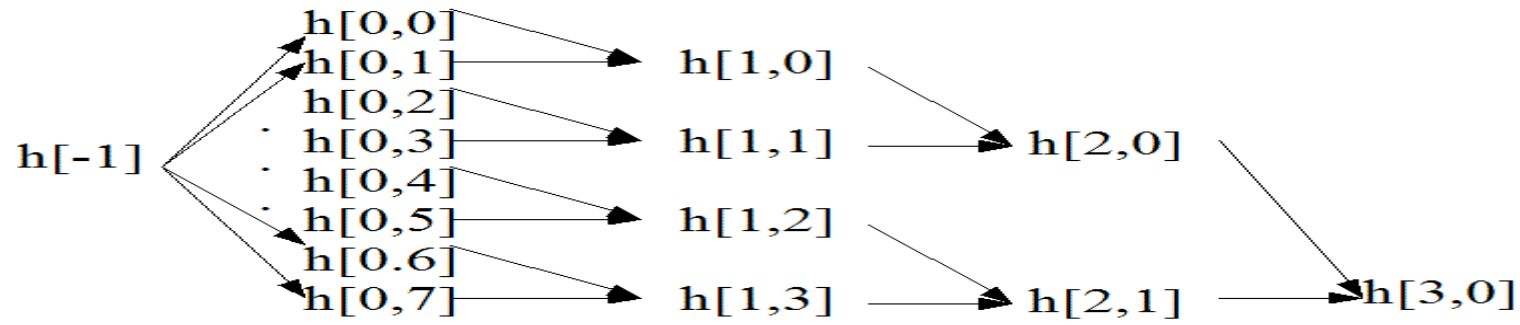
- Precomputation: Start with possible prefixes.
- No matching step, so this can be much cheaper!
- Example: Predict next big terrorist attack
  - $2^8=256$ locations (some specific, some general)
  - $2^{11}=2048$ days (covers >5 years)
  - $2^5=32$ descriptions (some specific, some general)
- Total of 2^{24} prefixes
  - 2^{77} work (128-bit hash)
  - All work on precomputation; "matching" is free!

15

Variation: Predicting an ordering
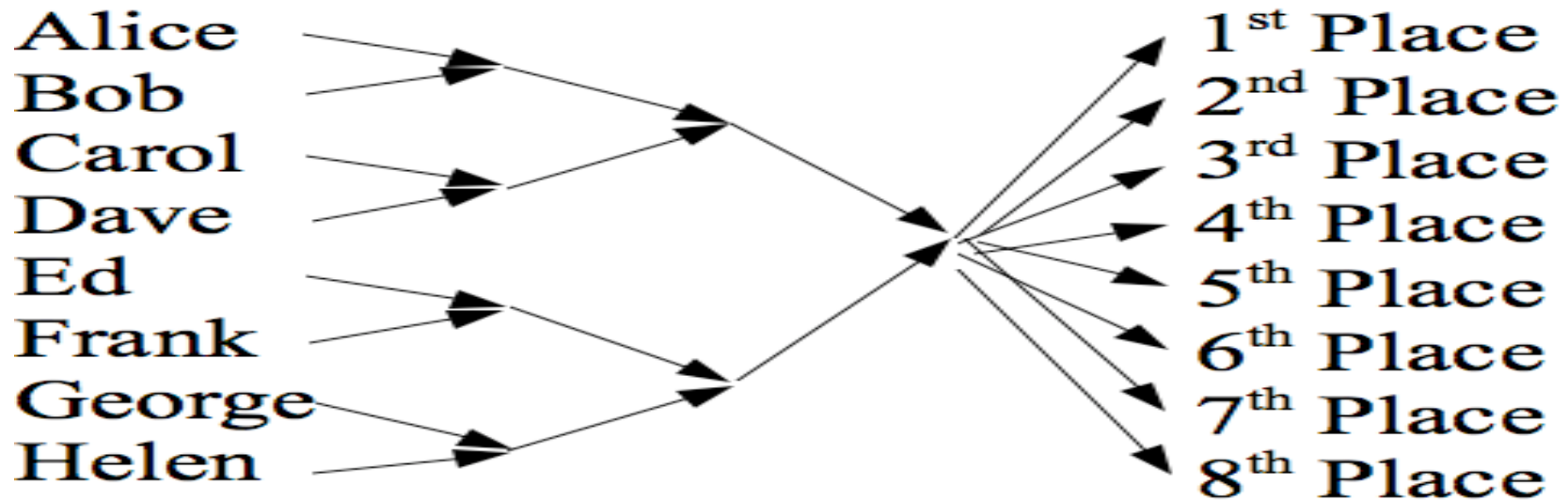
Predict order in which contestants will finish race

- Build diamond structure from contestant names
  – For 2^k contestants, $2^{n/2+k/2+3/2}$ total work.
- Append 8 message blocks to get 8 diff. hashes
- Commit to resulting hashes in order.

16

Routing "Bob" to "5th Place"

- Each arrow is a message block
  - Choose sequence of arrows to get Bob to 5th place
- Do same for each contestant to get full ordering
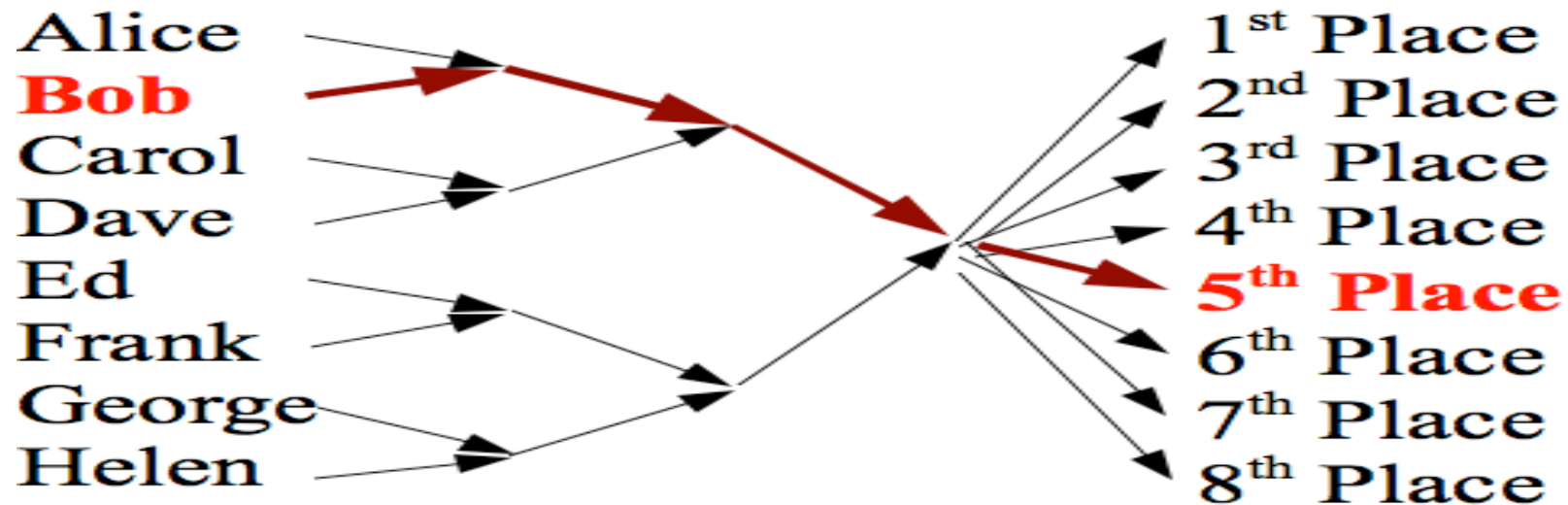
Using Joux Multicollisions

- "Which celebrities will marry this year?"
  - Sequence of yes/no answers
  - We can construct a Joux multicollision
  - Precomputation (MD5) : 2^{65} x R for
    R celebrities.
- Can reveal answers immediately at end of year
  - In this list, Alice and Dave got married....

Conclusions

- If I can find collisions on a hash function, it's **hard** to use securely

- Herding attack means a hash for which I can find collisions can't be used to commit to anything, prove knowledge, etc.

- Joux multicollisions even more powerful for special forms of commitment.

Herding Hash Functions and the Nostradamus Attack–DRAFT

John Kelsey[1] and Tadayoshi Kohno[2]

[1] National Institute of Standards and Technology, john.kelsey@nist.gov
[2] CSE Department, UC San Diego, tkohno@cs.ucsd.edu

Abstract. In this paper, we develop a new attack on Damgård-Merkle hash functions, called the *herding attack*, in which an attacker who can find many collisions on the hash function by brute force can first provide the hash of a message, and later "herd" any given starting part of a message to that hash value by the choice of an appropriate suffix. We introduce a new property which hash functions should have–Chosen Target Forced Prefix (CTFP) preimage resistance–and show the distinction between Damgård-Merkle construction hashes and random oracles with respect to this property. We describe a number of ways that violation of this property can be used in arguably practical attacks on real-world applications of hash functions. An important lesson from these results is that hash functions susceptible to collision-finding attacks, especially brute-force collision-finding attacks, cannot be used to prove knowledge of a secret value.

1 Introduction

Cryptographic hash functions are usually assumed to have three properties: Collision resistance, preimage resistance, and second preimage resistance. And yet many additional properties, related to the above in unclear ways, are also required of hash function in practical applications. For example, hash functions are sometimes used in "commitment" schemes, to prove prior knowledge of some information, priority on an invention, etc. When the information takes on more than a small number of possible values, collision resistance does not seem to be necessary to use the hash function in this way. This appears fortunate, in light of the many recent attacks on collision resistance of existing hash functions[BC04,RO05,Kli05,WLF$^+$05,WY05,BCJ$^+$05,WYY05b,WYY05a] and the widespread use of hash functions short enough to fall to brute-force collision attacks[vOW99]. But is collision resistance somehow necessary in this application?

1.1 Example: Proving Prior Knowledge with a Hash Function

Consider the following example. One day in early 2006, the following ad appears in the *New York Times*:

> I, Nostradamus, hereby provide the MD5 hash H of many important predictions about the future, including most importantly, the closing prices of all stocks in the S&P500 as of the last business day of 2006.

A few weeks after the close of business in 2006, Nostradamus publishes a message. Its first few blocks contain the precise closing prices of the S&P500 stocks. It then continues with many rambling and vague pronouncements and prophecies which haven't come true yet. The whole message hashes to H.

The main question we address in this paper is whether this should be taken as evidence that Nostradamus really knew the closing prices of the S&P500 many months in advance. MD5 has been the subject of collision attacks, and indeed is susceptible to brute force collision attacks, but there are no known preimage attacks. And yet, it seems that a preimage attack on MD5 would be necessary to allow Nostradamus to first commit to a hash, and then produce a message which so precisely described the future after the fact.

1.2 Chosen Target Forced Prefix (CTFP) Preimage Resistance

A natural question when considering the situation outlined above is what property of a hash function would have to be violated by Nostradamus, in order to falsely "prove" prior knowledge of these closing prices. The property is not directly one of the commonly discussed properties of hash functions (collision resistance[3], preimage resistance, and second preimage resistance). Instead, we need a new property, which we will call "chosen target forced prefix" (CTFP) preimage resistance.

In order to falsely prove his knowledge of the closing prices of the S&P500, Nostradamus would first have to choose a target hash value, H. He then would have to wait until the closing values of the S&P500 stocks for 2006 were available. Finally, he would have to find some way to form a message that started with a description of those closing values, P, and ended up with the originally committed-to hash H.

Following this example, we can formally define CTFP preimage resistance as follows: In the first phase of his attack, Phase1, Nostradamus performs some precomputation and then outputs an n-bit hash value H; H is his "chosen target". The challenger then selects some prefix P and supplies it to Nostradamus; P is the "forced prefix". In our security definition we place no restriction on how the challenger picks P, but for simplicity we may assume that the challenger picks P uniformly at random from some large but finite set of strings. In his second phase, Phase2, Nostradamus computes and outputs some string S. Nostradamus compromises the CTFP preimage resistance of the hash function if $\mathsf{hash}(P\|S) = H$. If we model the hash function as a random oracle, then unless Nostradamus is lucky and guesses P in Phase1, we would expect him to have to try $O(2^n)$ values for S in Phase2 before finding one such that $\mathsf{hash}(P\|S) = H$.

---

[3] Collision resistance would preclude the attack, but does not appear to be necessary for the attack to fail.

Consequently, it seems reasonable to expect that Nostradamus would have to perform $O(2^n)$ hash function computations to compromise the CTFP preimage resistance of a real hash function.

As described in detail below, the ability to violate the CTFP preimage resistance property allows an attacker to carry out a number of surprising attacks on applications of a hash function. Almost any use of a hash function to prove knowledge of some information can be attacked by someone who can violate this property. Many applications of hashing for signatures or for fingerprinting some information which aren't vulnerable to attack by straightforward collision-finding techniques are broken by an attacker who can violate CTFP preimage resistance.

Further, when the CTFP definition is relaxed somewhat (for example, by allowing Nostradamus some prior limited knowledge or control over the format of P, giving him prior knowledge of the full (large) set of possible P strings that might be presented, or allowing him to use any of a large number of encodings of P with the same meaning), the attacks become still cheaper and more practical.

1.3 Herding Attacks

The major result of this paper is as follows: For Damgård-Merkle[Dam89,Mer89] construction hash functions, CTFP preimage resistance can always be violated by repeated application of brute-force collision-finding attacks. More efficient collision-finding algorithms for the hash function being attacked may be used to make the attack more efficient, if the details of the collision-finding algorithms support this. An attack that violates this property effectively "herds" a given prefix to the desired hash value; we thus call any such attack a "herding attack."

The herding attack shows that the CTFP preimage resistance of a hash function like MD5 or SHA1 is ultimately limited by the collision resistance of the hash function. At a high level, and in its basic variant, the attack is parameterized by some positive integer k, e.g., $k = 50$, and by the output size n of the hash function. In Phase1 of a herding attack, the attacker, Alice, repeatedly applies a collision-finding attack against a hash function to build a *diamond structure*, which is a data structure reminiscent of a binary tree. With high probability it takes at most $2^{k/2+n/2+2}$ applications of the hash compression function (and possibly fewer, depending on details of more efficient collision-finding attacks[4] to create a diamond structure with $2^{k+1}-2$ intermediate hash states, of which 2^k are used in the basic form of the attack. In Phase2 of the attack, Alice exhaustively searches a string S' such that $P\|S'$ collides with one of the diamond structure's intermediate states; this step requires trying $O(2^{n-k})$ possibilities for S'. Having

---

[4] The collision finding attacks needed for constructing the diamond structure are somewhat different than those in recent results on MD5, SHA0, and SHA1[WY05,WYY05a]. We are uncertain whether these attacks can be adapted to the requirements of constructing the diamond structure, though it seems plausible that it might work. For the diamond structure we need collisions between two messages starting with different IVs.

found such a string S', Alice can construct a sequence of message blocks Q from the diamond structure, and thus build a suffix $S = S'\|Q$ such that $\mathsf{hash}(P\|S) = H$; this step requires a negligible amount of work, and the resulting suffix S will be $k+1$-blocks long. We stress that Alice can have significant control over the contents of S, which means that S may not be "random looking" but may instead contain structured data suitable for the application that Alice is trying to attack. We call our attack a "herding" attack because we use the diamond structure to "herd" the hash output of a given prefix to the previously-committed-to value H.

Table 1. Herding with Short Suffixes

output size	example	diamond width(k)	suffix (blocks)	work
128	MD5	41	48	2^{87}
160	SHA1	52	59	2^{108}
192	Tiger	63	70	2^{129}
256	SHA256	84	92	2^{172}
512	Whirlpool	169	178	2^{343}
n		$(n-5)/3$	$k + \lg(k) + 1$	2^{n-k}

Table 2. Herding with Impractically Long Suffixes

output size	example	diamond width(k)	suffix (blocks)	work
128	MD5	5	2^{55}	2^{69}
160	SHA1	15	2^{55}	2^{91}
192	Tiger	26	2^{55}	2^{112}
256	SHA256	47	2^{55}	2^{155}
512	Whirlpool	133	2^{55}	2^{325}
512	Whirlpool	5	2^{246}	2^{251}
n		$(n-2r-3)/3$	2^r	2^{n-k-r}

1.4 Practical Impact

Our techniques for carrying out herding attacks have much in common with the long message second preimage attacks of [KS05]. However, those attacks required implausibly long messages, and so probably could never be applied in practice. By contrast, our herding attacks require quite short suffixes, and appear to be practical in many situations. Similarly, many recent cryptanalytic results on hash functions, such as [WY05,WYY05a], require very careful control over the

format of the messages to be attacked. This is not generally true of our herding attacks, though more efficient variants that make use of cryptanalytic results on the underlying hash functions will naturally have to follow the same restrictions as those attacks.

Near the end of this paper, we describe a number of ways in which our herding attacks and variations on them can be exploited. We also describe closely-related attacks that can be done against signature protocols using our herding techniques, but not the more standard collision finding techniques.

In developing the herding attack, we also describe a new method of building multicollisions for Damgård-Merkle hash functions which is of independent interest, and which may be useful in many other hash function attacks.

1.5 Related Work

The herding attack is closely related to the long message second preimage attacks in [KS05] and [Dea99], and is ultimately built upon the multicollision-finding technique of [Jou04]. Our result complements Coron, Dodis, Malinaud, and Puniya's work[CDMP05] which does not present actual attacks like the ones we present, but shows that iterative hash functions like MD5 and SHA1 are not random oracles, even when their compression functions are. Our attack works against one of Coron, et al's fixes, but does not violate their provable security bound. In this particular case, our results can be seen as a tightness result.

More broadly, our result re-enforces the lessons that might sensibly be taken from [Jou04,KS05,Kam04,LWdW05,DL05] on the many ways in which seemingly impractical hash function collisions may be applied in practice. The security properties of Damgård-Merkle hash functions against attackers who can find collisions are currently not well understood.

1.6 Guide to the Paper.

The remainder of this paper is organized as follows: First, we describe the herding attack, and how it may be implemented. Next, we describe some techniques for enhancing the herding attack in plausible attack scenarios. We then discuss a number of arguably practical attacks which can be carried out using herding attacks, as well as some curiosities made possible by them. We conclude with lessons from the analysis and some open questions.

2 The Herding Attack: An Overview

The herding attack allows an attacker to commit to the hash of a message she doesn't yet fully know, at the cost of a large computation. This attack is closely related to the long message second-preimage attacks of [Dea99,KS05] and the multicollision-finding techniques of [Jou04].

At a high level, the attack works as follows:

1. The attacker, Alice, produces a large structure consisting of many possible intermediate hash values, and message blocks necessary to link all the values to a final hash output H. Note that the final hash output must incorporate a final padding block with a message length large enough to encompass the (still unknown) prefix and the linking message. She somehow commits to H.
2. Later, Alice gains knowledge of P. She then searches for a "linking message" M_{link} such that the intermediate hash value after processing $P||M_{link}$ is one of the intermediate hash values which appears in the attacker's structure.
3. Finally, she produces a sequence of message blocks from her structure to link this intermediate hash value back to the previously sent H.

In the long-message second preimage attack, the attacker has no control over the target message. The target message does, however, provide many different intermediate hash values, and a sequence of message blocks that will map each of them to the final hash value. The attacker produces an "expandable message," then searches for a "linking message" from the end of the expandable message to any of those intermediate states.

The herding attack is almost the long-message second preimage attack in reverse: The attacker chooses most of the target message, then has to carry out a second preimage attack against his own message from some previously-unknown prefix message P. The ability to choose most of the target message gives the attacker an enormous advantage, because she can build structures of connected message blocks and hash values which are much more efficient (in terms of the required length of messages relative to the number of reachable hash values) than a single long message.

This makes an enormous difference in the attacks that are possible, as well as the attacks' flexibility.

3 Building Structures of Messages: The Diamond and Elongated Diamond Multicollisions

For the herding attack, the attacker must produce a large set of intermediate hash values that can be reached in a legitimate message, as described above. In this section, we provide two ways to do this.
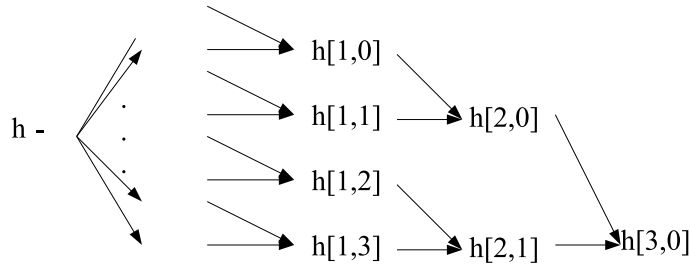
3.1 The Diamond Structure

In a second preimage attack, the attacker can't control the message. In the herding attack, he controls the "target message." This allows the attacker to trade work for suffix length in a very powerful way, by organizing a set of multicollisions into a tree structure. Figure 3.1 describes the basic idea: Note that edges represent messages, and values like $h[i,j]$ represent intermediate hash states. In the diagram, the attacker starts with eight different first message blocks, each leading to a different hash value; he then searches for collisions between pairs of these hash values, yielding four resulting intermediate hash values (at the cost

of about $8 \times 2^{n/2}$ work). He repeats the process with the four remaining values, then the two remaining ones. The result is that a diamond structure which is 2^k states wide, and contains $2^{k+1} - 1$ states total. If he fixes the tree structure prior to searching for collisions, then building the diamond structure costs about $2^{n/2+k+1}$ compression function computations to construct. But he can do much better: If he dynamically builds the tree structure during the collision search, then at each step in the search, when a collision is found between two hash values, they are put together as colliding values in the tree. Doing so makes building the diamond structure enormously cheaper, on the order of about $2^{n/2+k/2+2}$ compression function computations to construct.

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 line. Intuitively, we can make the following argument: (The current formula we have is mostly derived from experimental data with smaller n and k; the full paper will contain a more precise derivation of the formula.) When we try $2^{n/2+k/2+1/2}$ messages spread out over 2^k lines, we get $2^{n/2+k/2+1/2-k}$ messages per line, and thus between any pair of lines, 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 $2^{-k+k+1} = 2^1 = 2$ lines to collide with each line.

The diamond structure thus costs about $2^{k/2+n/2+1/2}$ steps to map 2^k lines down to 2^{k-1} lines. A sequence of calls to map 2^k lines down to one line thus costs about $2^{n/2+k/2+2}$.

Work for Herding Attacks with the Diamond Structure. A maximally short suffix for the herding attack is found by only searching for linking messages to the outermost (widest) level of hash values in the diamond structure, so that no expandable message is needed. In this case, the length of the suffix is $k + 1$ message blocks, and the work done for the herding attack is approximately

$$2^{n-k} + 2^{n/2+k/2+2} \tag{1}$$

Adding an additional $\lg(k)+1$ message blocks for a $(\lg(k), k+lg(k))$-expandable message[KS05] decreases the work required to

$$2^{n-k-1} + 2^{n/2+k/2+2} + k \times 2^{n/2+1} \tag{2}$$

The cheapest herding attack with a reasonably short suffixes can be determined by setting the work done for constructing the diamond structure and

finding the linking message equal. We thus get a diamond structure of width 2^k, suffix length L, and total work W, where:

$$k = \frac{n-5}{3} \tag{3}$$

$$L = \lg(k) + k + 1 \tag{4}$$

$$W = 2^{n-k-1} + 2^{n/2+k/2+2} + k \times 2^{n/2+1} \approx 2^{n-k} \tag{5}$$

Thus, using a 160-bit hash function, the cheapest attack with a reasonably short suffix involves a diamond structure with about 2^{52} messages at its widest point, producing a 59-block suffix, and with a total work for the attack of about 2^{109} compression function calls.

3.2 The Elongated Diamond Structure

Long messages offer a naive way to mount the attack, and the diamond structure offers much shorter suffixes. However, the attacker can make the attack much less expensive by using a very long suffix, as figure 3.2 shows. The widest layer of the diamond structure is chosen, with 2^k hash values. Then, the attacker computes 2^r message blocks for each of the 2^k hash values, thus producing a total of 2^{k+r} reachable intermediate states. He then constructs the collision tree as described above.
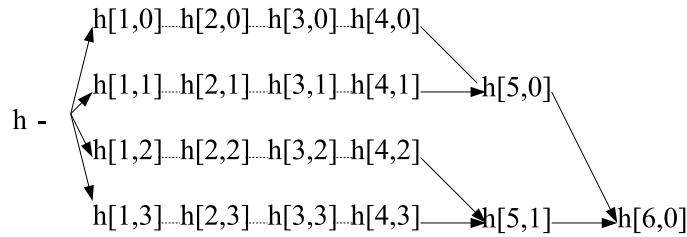
Work for Herding Attacks with the Elongated Diamond Structure
The cheapest herding attack with a suffix of slightly more than 2^r blocks can be determined by once again setting the work done for constructing the diamond structure and finding the linking message equal, so long as $k + r < k/2 + n/2$. We thus get an elongated diamond structure of width 2^k, suffix length L, and total work W, where:

$$k = \frac{n - 2r - 3}{3} \tag{6}$$

$$L = \lg(k + 2^r) + k + 1 + 2^r \tag{7}$$

$$W = 2^{n-k-r} + 2^{n/2+k/2+2} + k \times 2^{n/2+1} + 2^{k+r} \approx 2^{n-k-r+1} \tag{8}$$

Thus, with a 160-bit hash function and a 2^{54} block suffix (about as long as is allowed for SHA1 or RIPEMD-160), an attacker would end up doing about 2^{90} work total to herd any prefix into the previously published hash value.

4 Techniques and Variations for Using the Attack in the Real World

Above, we have defined what a herding attack is and broadly, how it may be done. In this section, we discuss tricks to make it more practical to actually use. Because of the very reasonable sizes of the suffixes in these attacks, and the very real dangers of collision-finding attacks on so many widely used hash functions, these attacks might really work; here we investigate how.

4.1 Using Yuval's Trick to Make Messages Meaningful

In all these attacks, we are choosing a bunch of message blocks to herd the current message back to some chosen hash output. Using Yuval's clever trick[Yuv79], the attacker can prepare a basic long document appropriate to her intended deception, and produce many independent variation points in the document. This allows the use of meaningful-looking messages for most contexts. For example, each message block in layer i of the diamond structure could be a variation on the same theme, using about $n/2$ possible variation points. The attack works in the same way if multiple message blocks are used per layer of the diamond structure, to accommodate a more restricted message space, though this naturally results in a longer suffix.

The contents of these suffixes must be pretty general. The natural way to handle this in most applications of herding is to write some common text discussing how the results are supposed to have been obtained ("I consulted my crystal ball, and spent many hours poring over the manuscripts of the ancient prophets...."). These can then be varied at many different points, independently, to yield many possible bitstrings all yielding the same (fraudulent) message.

4.2 Committing to Message Content vs. Bit Strings

For many of the attacks for which herding is useful, the goal is to falsely commit to some actual message content, not necessarily some specific message string. For example, an attacker trying to prove her ability to predict the stock market is not really forced to use any fixed format for the contents of her stock market predictions, so long as anyone reading them will unambiguously be able to tell whether she got her predictions right.

This provides a great deal of extra flexibility for the attacker in using Yuval's trick, and also in arranging the different parts of the message to be committed to, in order to maximize her convenience.

4.3 Using Joux Multicollisions in the Deception

In this section, we describe a similar, though limited, approach to the same kinds of attacks as are made possible by the herding attack, using only Joux multicollisions.

The attacker wishes to publish a hash that "commits" her to a value she doesn't yet entirely know. In the full herding attack, she might well know nothing about it. In this attack, she knows how to construct the unknown message out of pieces she can define ahead of time and include in a Joux multicollision. That is, she is going to produce a long message consisting of independent pieces, where she has a choice for the value of each piece.

Consider the case where the attacker wishes to commit to a sequence of success or failure type events, without knowing them. That is, each event can be described ahead of time, and must yield either a failure or a success. An example of this would be a list of famous people who will or will not marry during the year.

The attacker chooses her list of famous people to include. For each person, she constructs a scheme for generating about $2^{n/2}$ different variations on the two basic messages "This person will get married this year" and "This person will not get married this year." The full messages must take up an integer number of blocks, and it must be the same number of blocks for both the "yes" and "no" answer.

The attacker then constructs a Joux multicollision, in which the ith collision is between one message that says that person i will marry next year, and another that says they will not. When finished, she publishes the hash of her predictions for the future.

At the end of the year, she "reveals" her predictions, choosing for each pair of colliding blocks the one that reflects what did happen that year.

The result is somewhat like the herding attack, though it it far more constrained. However, it has three important advantages:

1. The attack requires less work.
2. The attack frontloads the work; on Jan 1, 2006, the attacker can reveal her answers immediately, without an expensive computation.
3. Within the constraints of making plausible-looking messages, the attack can make direct use of existing attacks on hash functions such as MD5 and SHA1 since, for each collision search, the IVs are the same.

Further, this technique can be combined with the more general herding attack to provide even more flexibility, as we will discuss below.

4.4 Precomputing All Possible Prefixes

In the herding attack, the attacker may reasonably expect to produce a diamond structure with 2^{50} or more possible hash values. For a great many possible applications of the herding attack, this may be more than the possible number of prefix messages. The attacker may now take advantage of an interesting feature of the diamond structure: there is no restriction on the choice of starting hash values for the structure.

Let 2^k, the width of the diamond structure, be the number of possible prefix messages that the attacker may need to herd to her fixed hash value. (If there are

fewer prefix messages, the attacker append one block to all the possible prefix messages, and vary that block to produce a set of prefix messages that is exactly the right size.) She computes the intermediate hash after processing each prefix message, and uses these intermediate hashes as the starting hash values for the diamond structure.

The initial work to construct the diamond structure in this way is the same as for the more general herding attack. However, the attacker now has the ability to immediately produce a message which starts with any possible prefix with the desired hash value. That is, she need not do a second expensive computation to herd the prefix she is given.

The attacker who has a larger set of possible prefixes than this is not lost; she may precompute the hashes of the most likely 2^k prefixes. Then, if any of those prefixes is presented to her, she can herd it immediately; otherwise, she must do the large computation.

4.5 Combining Precomputations and Joux Multicollisions

In some cases, some large part of the prefix messages will be precomputable, but not everything. For example, a hash which commits to a high-level summary of the outcomes of the 2008 federal elections has a very small set of possible kinds of content, in the sense that either major party may win heavily, or may win by a narrow margin, or neither may win a clear victory, but it's vanishingly unlikely that a third party candidate will win, or that a list of the 100 most likely people to run on either side compiled today by an interested observer would fail to list the two major-party presidential candidates. On the other hand, a listing of which party carried each state's electoral votes for president might have too many possibilities to be entirely predicted, and a county-by-county summary certainly would be. However, the Joux multicollision trick can be used there; the message is constructed so that it starts with a terse statement about which party won the presidency, then a summary of who won in each state or county, and then the text discussion summarizing the election. Because the two parties and the summary of which party won in each state can be put together using a Joux multicollision, the attacker can still precompute all possible prefix messages, and ensure that she can quickly produce a message that "proves" she had predicted to whatever happens in the election.

4.6 Applying the Joux Multicollision Idea to Diamond Structures

In some cases, the possible strings Alice may need to produce may not quite fit the Joux multicollision pattern, but *almost* fit. For example, a string describing how the 2008 election will come out in each state would lose a great deal of specificity if it committed only to "Democrat/Republican." An alternative is for Alice to build a small precomputed diamond structure for each state, encompassing all plausible outcomes. For example, ignoring third party votes, Alice could build a precomputed diamond structure committing to (82/18,81/19,...,19/81), a set of

64 possible descriptions which covers essentially every possible outcome, for each state. After the polls closed, she could then reveal her "uncanny" predictions.

This small precomputed diamond could be built for each state, allowing Alice to instantly reveal her "predictions" with uncanny accuracy, even though she is specifying a great deal of detailed information.

4.7   The Simultaneous Multicollision

The diamond structure used in the attack lends itself to finding a large number of different messages that all have the same hash. It's possible to have a protocol where Alice sends out a hash H, and then Bob sends many prefixes $P_{0,1,...,R-1}$, and for each one, Alice responds with a suffix S_i such that $\mathsf{hash}(P_i||S_i) = H$.

4.8   The Hash Router

A large diamond structure can be constructed, leading to a hash value H. A large number of different input blocks may be processed from that hash value, to get to a large number of hash outputs. Now, any given prefix can be herded to any of these specified hash outputs.

5   Applying the Attack: Herding for Fun and Prophets

In this section, we describe how the herding attack can be used in many different contexts to do surprising things.

5.1   The Nostradamus Attack

We first return to Nostradamus:

One day in early 2006, the following ad appears in the *New York Times*:

> I, Nostradamus, hereby provide the MD5 hash H of many important predictions about the future, including most importantly, the closing prices of all stocks in the S&P500 as of the last business day of 2006.

A few weeks after the close of business in 2006, "Nostradamus" publishes a message. Its first few blocks contain the precise closing prices of the S&P500 stocks. It then continues with many rambling and vague pronouncements and prophecies which haven't come true yet. The whole message hashes to H.

Discussion. The Nostradamus attack is carried out in order to convince people that the attacker can tell the future. This could be based on some claimed psychic power, but also on some claimed improved understanding in science or economics, allowing detailed prediction of the weather, elections, markets, etc. This can also be used to "prove" access to some inside information, as with some attacker attempting to convince a reporter or intelligence agent that she has inside access to a terrorist cell or secretive government agency.

At a very general level, this attack works as follows:

1. The attacker presents the victim with a hash H, along with a claim about the kind of information this represents. She promises to produce the message that yields the hash after the events predicted have occurred.
2. The attacker waits for the events to unfold, just as the victim does.
3. The attacker herds a description of the events as they did unfold into her hash output, and provides the resulting message to the victim, thus "proving" her prior knowledge.

In many cases, the possible events to be committed to may be precomputed, allowing the attacker to "prove" her prior knowledge as soon as the events "predicted" unfold, with no further expensive computation.

Variation: Stealing Credit for Inventions. The attacker can use the same idea to claim to be a brilliant inventor, while actually stealing other peoples' work. He submits hashes to a digital timestamping service periodically. After he sees some new invention he wants to claim, he herds a description of the invention to some old hash value.

This attack uses the "hash router" structure: a diamond structure with a single additional message block after it. Because the attacker must send many hashes, but need only herd one message to the right value (the one which shows the attacker's prior claim to the invention), the attacker must vary the final message block after the diamond structure for each hash sent.

5.2 Committing to an Ordering

Alice decides to prove (perhaps in a gambling context) that she can predict the outcome of a race with 32 entrants. She commits to a sequence of 32 hash outputs, $H_{0,1,...,31}$. After the race is over, she produces 32 strings, $S_{0,1,...,31}$ such that S_i describes the entrant in the race who finished in ith place, and $H_i = \mathsf{hash}(S_i)$.

This attack uses the "hash router," above. Alice builds a diamond structure starting from the names of the 32 entrants. When the diamond structure yields a final hash H, she produces 32 new message strings (probably simply strings like "1st place", "2nd place", etc.), and processes them from H to get 32 different hash outputs. She commits to these hash outputs. When the time comes to reveal her choices, she produces 32 strings which commit her to the correct ordering of entrants in the race.

5.3 Editing Messages Without Changing the Hash

Above, we have described the herding attack as applying when the victim chooses a *prefix*, but the real attack is richer than this–it applies when an attacker has control over the last few blocks of any message. Thus, the attacker or some innocent party can provide a long file, the text or HTML version of a newspaper or textbook, etc. She can determine a hash for it, digitally sign the hash, and get

it timestamped. She can then get others to *edit* the message anywhere but in its last 55 or so blocks. That is, the attacker can allow news stories in the HTML version of the newspaper to be rewritten, can alter discussions, etc. Whenever she decides to, she can herd this back to the already-published hash by altering only those last 55 or so blocks.

This defeats the use of hashes to "freeze" information so that it can't later be altered. Further, unlike a straightforward collision attack, which would allow an attacker to find two different versions of the information with the same hash, this attack allows the attacker to edit the data at will. Even more interestingly, it allows the attacker to incorporate innocent parties' edits of the information.

How it Works. Alice constructs a diamond structure first. She then chooses one path through the diamond structure. Whenever the victim edits the message, the attacker simply alters the last message block before the diamond structure to link back into the diamond structure in a different place.

Moving the Herding Blocks Around. Above, we have assumed that the herding blocks had to appear at the end of the message. However, this isn't really the case. The attacker may append fixed message blocks to the end of her diamond structure, so that while she must change 55 or so successive message blocks at some point in the message after the last edit, she need not change the last 55 blocks. However, this still requires that the attacker control or at least know the last 55 blocks of the original message, and also of all edited variations.

However, a bit of cleverness on the attacker's part can deal with some very small amount of variation in those last few blocks. Consider the situation in which the victim will decide, after the original message and hash are committed to, which of two possible suffix strings,S_0 or S_1, is to be appended to the end. The attacker can still deal with this; she appends one message block after her diamond structure, and computes about $2^{n/2}$ choices for that message block. For each possible one, she computes the final hash result after processing both S_0 and S_1. By the birthday paradox, she expects to find a pair of message blocks Z_0, Z_1 such that $\mathsf{hash}(...||Z_0||S_0) = \mathsf{hash}(...||Z_1||S_1)$. With a rapidly-increasing amount of work, she can extend this to deal with small numbers of alternative S_i. (For example, an attacker trying to find 10 messages with the same 160-bit hash expects to have to try about 2^{144} message blocks.)

5.4 Overcoming Code Review: Herding Messages Created with an Innocent Party

The herding attack is useful (to an attacker, anyway) even when nobody is being fooled about prior knowledge or commitment to a bitstring. Consider the situation in which a voting system vendor submits source code to a testing lab, and must go through many rounds of comments and updates to the source code before finally passing the evaluation. Further suppose that the testing lab *always*

requires some unpredictable-to-the-programmer changes as a way of making it more difficult to insert intentional bugs in the system.

The attacker has taken over the voting system company, and now does a large precomputation to get a diamond structure with output hash H, which she does not publish. She submits her initial source code with only the certainty that she can alter the final few hundred bytes of some source code file; perhaps the end of the file is filled with freeform documentation of recent changes. She goes through the process of submitting the code, getting required changes, etc. Each time, before she submits the source code file being attacked, she alters the last few hundred bytes to herd the hash to her chosen value. When the testing lab finally passes her source code, it appends a digital signature to each source code file, to ensure election officials using the voting system that they are getting properly reviewed source code.

The attacker now has a source file which she can edit later without changing the hash, despite the fact that the file was created by an interaction with a trustworthy entity. Thus, she can produce an altered version with a trapdoor included, and replace it in the signed distribution for the next election.

Note that the same basic idea could apply to any message being produced, such as object code, postscript, a text contract, etc.

5.5 Controlling the Full Message by Controlling a Suffix

Consider the following situation: Alice and Bob are working together on a large document; Bob is responsible for the first part, while Alice is responsible for the second. Bob (wisely) mistrusts Alice, and understands the recent hash results, so he insists that his part of the document go first. Bob produces the first part of the document, for which he is to be held responsible, D_B. Alice then produces her part, for which she is to be held responsible, D_A. They both hash and sign the complete document $D_B || D_A$.

Without the herding attack, Bob appears safe; Alice can certainly introduce collisions on her part of the document, but Bob is not held responsible for them, so why should he care?

With the herding attack, Alice chooses the last few blocks of D_A to herd the message into her diamond structure. She can then take the hash of this document, and alter Bob's section at will, without altering this hash.

5.6 Random Number Fixing

Alice and Bob want to agree on a shared random sequence for some game. Alice sends $\mathsf{hash}(X_1)$, then Bob responds with X_2. Finally, Alice reveals X_1, and Alice and Bob each derive random bits by combining X_1 and X_2 in some way. The herding attacks and its variations can be used to allow Alice to exert substantial control over the resulting random bit sequence.

Suppose Alice and Bob each contribute an equal-length message to the protocol, and that random bits are derived by XORing the two messages together.

A conventional use of a collision attack would give Alice only two choices for X_1. A Joux multicollision attack in this case gives Alice enormous flexibility–she can choose two possibilities for each message block. If random numbers are derived one message-block-sized chunk at a time, then Alice gets two choices for each random number generated, while Bob has no power at all over them.

A herding attack would allow Alice to be even more powerful in principle–she could choose *any* sequence of message blocks until the last 55 or so, which she would need to herd the X_1 she sent to the committed hash value. However, without some precomputation, Alice would have a very hard time herding her choices for X_1 to the value to which she had committed quickly enough for Bob to continue the protocol with her.

In this attack, the herding attack isn't used to prove prior knowledge, but rather to change a value after it has been committed to.

6 Other Applications of the Herding Attack and Diamond Structure

6.1 Multiblock Fixed Points

Attacks on commitment schemes are not the only applications of the diamond structure and herding attack ideas. We can also find short cycles in hash functions. This is done in a simple way: we first construct a diamond structure, where each of the starting hash values in the structure are found by generating a random message block, and computing the compression function result of that message block from the hash function's initial value. If the diamond structure is 2^k wide, we then compute 2^{n-k} trial message blocks from the end of the diamond structure. We expect an intermediate collision, which yields a k-block fixed point for the hash algorithm.

This can be extended; with 2^{n-k+r} work, we expect about 2^r different k-block fixed points, all reachable from a legitimate message. These can be concatenated together; we can choose which of the 2^r k-block chunks of message we wish to append to the message next, without reference to previous choices. Further, any message can be "herded" to this set of fixed points with about 2^{n-k} work and k appended blocks.

7 Conclusions

In this paper, we have defined a property of a hash function, Chosen Target Forced Prefix (CTFP) preimage resistance, which is both surprisingly important for real-world applications of hash functions, and also surprisingly dependent on collision resistance of the hash function. We have described a variation on the Joux multicollision technique for building tree-like structures of multicollisions called "diamond structures," and enumerated a number of techniques made possible by these structures. We have described a number of arguably practical attacks which use these techniques.

At a very basic level, the most important lesson the reader can take from this paper is that using hash functions whose collision resistance has been violated is very difficult, even when the relevant security property does not appear to depend on collision resistance.

A great deal of research remains to be done in this area. The diamond structure seems likely to us to be about as useful in developing new attacks as the Joux multicollision result, and we hope to see others building on the work in this paper by finding other surprising things to do to iterated hash functions using herding attacks and the diamond structure. Additionally, there may be many other surprising ways in which iterated hash functions built on the Damgård-Merkle construction may be attacked, when the attacker can find intermediate collisions.

8 Acknowledgments

References

[BC04]   E. Biham and R. Chen. Near-collisions of SHA-0. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 290–305. Springer-Verlag, Berlin, Germany, 2004.

[BCJ⁺05] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet, and W. Jalby. Collisions of SHA-0 and Reduced SHA-1. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[CDMP05] J.-S. Coron, Y. Dodis, C. Malinaud, and P. Puniya. Merkle-Damgård revisited: How to construct a hash function. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[Dam89]   I. Damgård. A design principle for hash functions. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer-Verlag, Berlin, Germany, 1989.

[Dea99]   R. D. Dean. *Formal Aspects of Mobile Code Security*. PhD thesis, Princeton University, January 1999.

[DL05]   M. Daum and S. Lucks. Attacking hash functions by poisoned messages: The story of Alice and her boss, 2005. `http://www.cits.rub.de/MD5Collisions`.

[Jou04]   A. Joux. Multicollisions in iterated hash functions. Application to cascaded constructions. In M. Franklin, editor, *Advances in Cryptology – CRYPTO 2004*, volume 3152 of *Lecture Notes in Computer Science*, pages 306–316. Springer-Verlag, Berlin, Germany, 2004.

[Kam04]   D. Kaminsky. MD5 to be considered harmful someday. Cryptology ePrint Archive, Report 2004/357, 2004. `http://eprint.iacr.org/`.

[Kli05]   V. Klima. Finding MD5 collisions on a notebook PC using multi-message modifications. Cryptology ePrint Archive, Report 2005/102, 2005. `http://eprint.iacr.org/`.

[KS05]   J. Kelsey and B. Schneier. Second preimages on n-bit hash functions for much less than 2^n work. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 474–490. Springer-Verlag, Berlin, Germany, 2005.

[LWdW05] A. Lenstra, X. Wang, and B. de Weger. Colliding X.509 certificates. Cryptology ePrint Archive, Report 2005/067, 2005. `http://eprint.iacr.org/`.

[Mer89]   R. C. Merkle. One way hash functions and DES. In G. Brassard, editor, *Advances in Cryptology – CRYPTO'89*, volume 435 of *Lecture Notes in Computer Science*, pages 428–446. Springer-Verlag, Berlin, Germany, 1989.

[RO05]   V. Rijmen and E. Oswald. Update on SHA-1. In A. Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 58–71. Springer-Verlag, Berlin, Germany, 2005.

[vOW99]   P. van Oorschot and M. Wiener. Parallel collision search with cryptanalytic applications. *Journal of Cryptology*, 12(1):1–28, 1999.

[WLF+05]   X. Wang, X. Lai, D. Feng, H. Chen, and X. Yu. Cryptanalysis of the hash functions MD4 and RIPEMD. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[WY05]   X. Wang and H. Yu. How to break MD5 and other hash functions. In R. Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, volume 3494 of *Lecture Notes in Computer Science*, pages 19–35. Springer-Verlag, Berlin, Germany, 2005.

[WYY05a] X. Wang, Y. L. Yin, and H. Yu. Finding collisions in the full SHA-1. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[WYY05b] X. Wang, H. Yu, and Y. L. Yin. Efficient collision search attacks on SHA-0. In V. Shoup, editor, *Advances in Cryptology – CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*. Springer-Verlag, Berlin, Germany, 2005.

[Yuv79]   G. Yuval. How to swindle Rabin. *Cryptologia*, 3(3):187–189, 1979.