Keying Hash Functions for Message Authentication

Hyrum Mills, Chris Soghoian, Jon Stone, Malene Wang

Basic Notions and Definitions: MACs:

- What do they do?
  - Allow one party to verify the integrity and authenticity of information transmitted over an insecure channel by another party, or of information stored in a medium that may be subject to modification
  - Usually require a secret key shared by both parties

Basic Notions and Definitions: MACs:

- What do they do?
  - X -> Y: (m, a)
  - where X and Y are the two parties, m is the message
  - a is called the authentication tag
  - a = $\mathrm{MAC}_k(m)$, which is the MAC function of message m using key k
  - Y takes m and computes a' = $\mathrm{MAC}_k(m)$
  - If a = a', then Y knows the message was not altered en route from X to Y

Basic Notions and Definitions: MACs:

- What does it mean to break one?
  - An adversary A sees a sequence of messages (m_i, a_i) (where i=1,2,...,q) between X and Y
  - A breaks the MAC if she can find a message m which does not equal any of m_1,...,m_q, and also a valid corresponding a = $MAC_k(m)$

Basic Notions and Definitions: MACs:

- Known Message Attacks (passive attack)
  - X and Y transmit the sequence of (m_i, a_i) pairs in a way uninfluenced by A, and A simply eavesdrops and picks up the pairs
- Chosen message attacks (active attack)
  - A can choose the sequence of (m_i, a_i) pairs.
  - We consider the chosen messages as "queries" sent by A and answered by X or Y or whomever knows the secret key k

Basic Notions and Definitions: MACS:

- Known/chosen message attacks
  - MACs that are secure against chosen messages are stronger than those that are secure against only known messages
  - Focus on chosen message attacks

Basic Notions and Definitions: MACS:

- A MAC is a (epsilon, t, q, L)-secure MAC if against adversary A if A:
  - is not given the key k
  - is limited to spend total time t (measured in number of operations) on the attacks which includes both the time taken for the answers to A's queries to be computed, and the size of the code of A's algorithm

Basic Notions and Definitions: MACS:

- A MAC is a (epsilon, t, q, L)-secure MAC if an adversary A that:
  - can make at most q queries on messages m_1, ..., m_q of her choice
  - each message is at most of length L

... can break the MAC with at most probability epsilon

Basic Notions and Definitions: Cryptographic hash functions

- What are they and what are they good for?
  - Cryptographic hash functions map strings of different lengths to short, fixed-sized outputs

Basic Notions and Definitions: Cryptographic hash functions

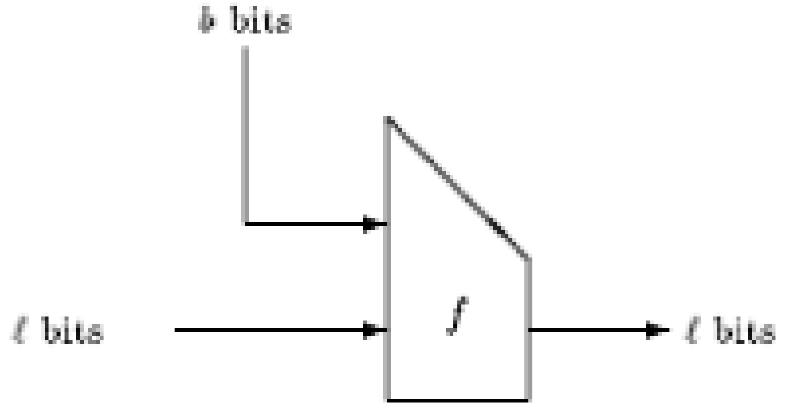
- Important Properties
  - Collision-resistant
    - Given a hash function F, it is infeasible for an adversary A to find two strings x and x' such that F(x) = F(x')
  - The function must be publicly computable and doesn't require a secret key

Basic Notions and Definitions: Cryptographic hash functions

- Important Properties
  - Randomness-like
    - Mixing, independence of input/output, unpredictability of the output when parts of the input are known, etc
    - Makes it more difficult to find collisions
    - Helps randomize the output

Basic Notions and Definitions: Iterated hash functions

- Examples: MD5 & SHA1



- Compression function *f*
- Accepts 2 inputs: *chaining variable* and *b*-bit data block

Basic Notions and Definitions: Iterated hash functions

Operation:

0) Pad the message to an exact multiple of b bits somehow

1) Fix the IV

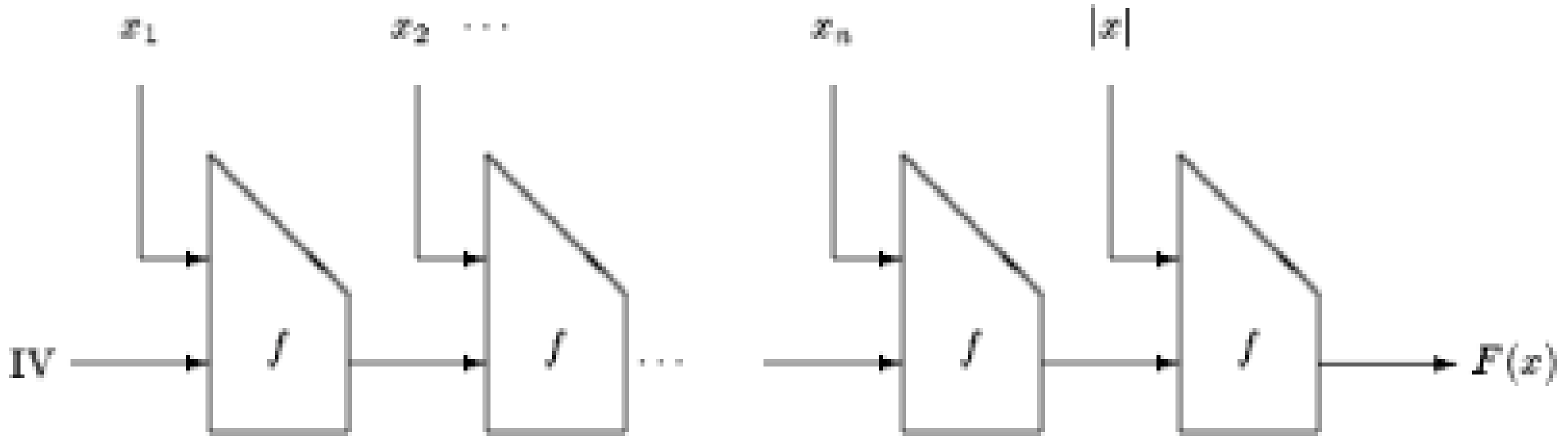
- *l*-bit value used as the first chaining variable

Basic Notions and Definitions: Iterated hash functions

2) Iteration:

- Let the input string is $x = x_1, ..., x_n$, where the x_i's are blocks of length b, and n is the number of blocks
- $|x|$ is the message length

Basic Notions and Definitions: Iterated hash functions



- The iterated function $F(x) = h_{n+1}$,where:

  $h_0 = IV$

  $h_i = f(h_{i-1}, x_i)$ for $i = 1, ..., n+1$

  $x_{n+1} = |x|$

Basic Notions and Definitions: Iterated hash functions

Example:

Iteration 1: $h_1 = f(IV, x_1)$

Iteration 2: $h_2 = f(h_1, x\_2)$

Iteration 3: $h_3 = f(h_2, x\_3)$

....

Iteration n+1: $h_{n+1} = f(h_n, |x|)$

Return $F(x) = h_{n+1}$

Basic Notions and Definitions: Keyed Hash Functions

- Keyed Hash Functions
  - As mentioned earlier, cryptographic hash functions are key-less by default.
  - The most common approach: $F_k(x) = F(k \| x)$
  - In this paper we use the key k as the IV, instead of using a known and fixed value as the IV

Basic Notions and Definitions: Keyed Hash Functions

- Keyed Hash Functions
  - Using the keyed IV approach, the definition of keyed hash functions:
    - Let f_k be the compression function, where $f_k(x) = f(k, x)$, $|k| = l$, and $|x| = b$
    - We associate to any iterated hash construction a family of keyed functions $\{F_k\}$
    - In particular, the original iterated hash function is a member of this family, F_{IV}

Basic Notions and Definitions: Keyed Hash Functions

- Iterated Functions:
- $F(x) = h_{n+1}$, where:

  $h_0 = IV$

  $h_i = f(h_{i-1}, x_i)$

  $x_{n+1} = |x|$

- Keyed Hash Functions:
- $F_k(x) = k_{n+1}$, where:

  $k_0 = k$

  $k_i = f_{k\_i-1}(x_i)$

  $x_{n+1} = |x|$

- Bad notation for keyed hash functions in the paper
- Notice that keyed hash functions are the same as the original iterated function but with an IV of *k*

Basic Notions and Definitions: Keyed Hash Functions

- A family of keyed hash functions $\{F_k\}$ is (epsilon, t, q, L) - weakly collision-resistant if adversary A:
  - is not given the key k
  - is limited to spend total time t (in comp. operations)
  - can see the values of F_k computed on at most q messages $m_1, ..., m_q$, which are chosen by A
  - each message is at most of length L
  - can find messages m and m' such that $F_k(m) = F_k(m')$ (in other words, find a collision) with at most probability epsilon

Basic Notions and Definitions: Keyed Hash Functions

This requirement is weaker than the traditional requirement of collision-resistance from key-less hash.

Basic Notions and Definitions: Keyed Hash Functions

- Key-less hash functions have a known and fixed IV
- Adversary can work on finding collisions in key-less hash functions without interacting with a user or knowing the key, which means that parallelization of brute force attacks are possible
- In contrast, attacks on keyed hash functions require the adversary to interact with the legal user by querying her with chosen messages, which means that parallelization is not possible

MACing with Cryptographic Hash Functions

- Advantages:
  - The popular hash functions are fast in software
  - These software implementations are widely and freely available
  - Hash functions are not subject to the export restriction rules of many countries

MACing with Cryptographic Hash Functions

- Caveat:
  - Hash functions were not designed for message authentication
  - One difficulty, in particular, is that they are key-less
  - Special care must be taken when using hash functions for MACs, as you are using them in a way they were not designed for

Nested Construction NMAC

- Definition of NMAC
  - Let $k = (k_1, k_2)$, where k_1 and k_2 are keys to the hash function F (i.e. random strings of length l each)
  - $NMAC_k(x) = F_{k\_1}(F_{k\_2}(x))$
  - The outer function is iterated only once, and is essentially just the compression function f_{k_1}

Nested Construction NMAC

- Performance
  - Cost of internal function: just the same as hashing the data with the basic key-less hash function
  - Additional cost is only one iteration of the external compression function

Nested Construction NMAC

- Security analysis
- Theorem 4.1:
  - If the keyed compression function f is an (epsilon_f, q, t, b)-secure MAC on messages of length b bits, and the keyed iterated hash F is (epsilon_F, q, t, L)-weakly collision-resistant, then the NMAC function is an (epsilon_f + epsilon_F, q, t, L)-secure MAC

Nested Construction NMAC

- Security analysis
- Theorem 4.1:
  - In other words, given the above resource constraints, an adversary attacking $NMAC_k$ has a probability of success P_s, where
  - $P_s \leq 2 * epsilon_f$
  - $P_s \leq 2 * epsilon_F$

Nested Construction NMAC

- Remarks 4.2:
  - The proof is *constructive*. Namely, given an adversary that breaks NMAC with some significant probability, an adversary can come up with an algorithm using the same resources that breaks the underlying hash function with at least half of that probability
  - The degradation of security, when going from the underlying hash function to NMAC, is minimal
  - Also, this proof considers a generic adversary, and takes into account all possible attackers

Nested Construction NMAC

- Remarks 4.3:
  - The definitions and analysis are stated in terms of chosen message attacks, which are the strongest
  - However, can refine everything to quantify separately the number of chosen and known messages in an attack
  - The analysis preserves the number of chosen and known messages when translating an attack on NMAC to an attack on the underlying hash

Nested Construction NMAC

- Remarks 4.7:
  - The security of the function is given by each individual key length (l), and not the combined length ($2l$)

HMAC: A Fixed IV Variant

Disadvantage of NMAC:

- The underlying hash must be modified to key the IV (not too difficult in software)
- Definition HMAC:
  - $HMAC_k(x) = F(\bar{k}$ XOR opad || $F(\bar{k}$ XOR ipad || $x))$
  - where $\bar{k}$ is k padded to a multiple of b bits
  - opad and ipad are two fixed b-bit constants.
  - opad is formed by repeating the byte 0x36 as often as needed, ipad by repeating the byte 0x5c

HMAC: A Fixed IV Variant

- Security analysis
- HMAC is a special case of NMAC
  - Recall that $NMAC_k = F_{k\_1}(F_{k\_2}(x))$
  - Define NMAC's two keys such that
  - $k_1 = f(IV, \bar{k}$ XOR opad)
  - $k_2 = f(IV, \bar{k}$ XOR ipad)

HMAC: A Fixed IV Variant

- Since the security of NMAC depends on the randomness of its keys, this requires that k_1 and k_2, derived from k using the function f, are indistinguishable from truly random keys.
- This requires an additional assumption as to how pseudorandom the function f is (MD5, SHA-1, etc. are already pretty pseudorandom)

HMAC: A Fixed IV Variant

- Security of HMAC vs. NMAC.
- The additional assumption: Not too stringent, since we require a relatively weak form of pseudo-randomness
- Theoretically, attacks that work on HMAC and not on NMAC are possible, but would imply major weaknesses in the underlying hash function

HMAC: A Fixed IV Variant

- In practice, k_1 and k_2 in NMAC are probably pseudo-randomly generated anyway
- The functions involving ipad and opad provide a "built-in" pseudorandom generator in HMAC

HMAC: A Fixed IV Variant

- Values of ipad and opad were chosen to:
  - Simplify function specification
  - Maximize Hamming distance between the pads to provide computational independence between the two derived keys
    - A Hamming distance is the number of bits that differ between two n-bit binary strings
- Implications of one key versus two:
- Using one L-bit long key instead of two doesn't weaken the function with respect to exhaustive key searches. See divide and conquer (later)

HMAC: A Fixed IV Variant

- Advantages of HMAC
  - Black box usage of underlying hash function
  - Requires only one l-bit long key, as opposed to two keys as in NMAC
- Implementation considerations
  - Slower performance than NMAC
  - Counter it by caching the keys (no more black box)
  - Implications of various key lengths
  - Crypto 101: key management is important

Advantages of NMAC/HMAC

- Generic attacks - proof and analysis considers generic attacks
- Exact analysis - no asymptotics involved

Advantages of NMAC/HMAC

- Tight relationship between security of NMAC/HMAC and the assumed strengths of the underlying hash function
- Insecurity of NMAC/HMAC => insecurity of underlying hash (more details to come when we go over the proof)
- Besides, NMAC/HMAC require significantly weaker properties, so it could be secure even if the underlying hash function were not

Advantages of NMAC/HMAC

- Efficient - minimal performance degradation caused by using a hash scheme
- If black box usage of hash functions is desired, use HMAC. Otherwise, modifying the hash function to cater to NMAC is also simple to do (in software, at least).

Attacks

- Birthday attacks
- Extension attacks
- Divide and conquer attacks

Birthday Attacks

Paradox

- 23 people in a group, there is slightly more than a 50% chance that at least two of them will have the same birthday. 60 or more people, the probability is greater than 99%.
- Not really a paradox

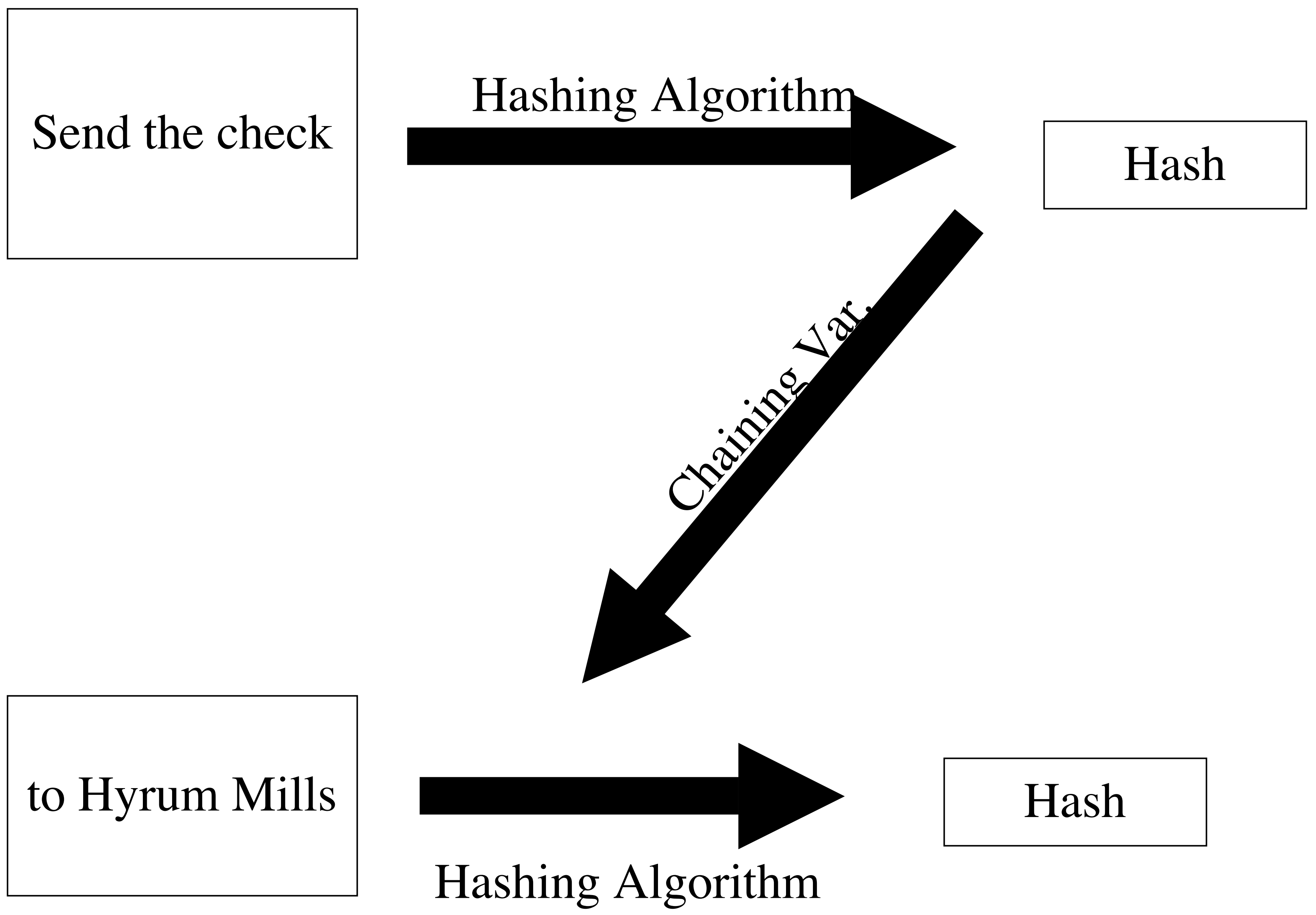
Attack

- Uses the mathematics underlying the paradox in order to minimize both the time and space required to run the attack.
- Math:
  - If a given function returns any of n unique outputs with equal probability, and n is a large enough value, then after running $1.2\sqrt{n}$ different values through the function we can expect to have found a pair of arguments x_1 and x_2 where $x_1 \neq x_2$ but $f(x_1) = f(x_2)$ – known as a collision.
  - If the function range is unevenly distributed, a collision may occur even faster.

Extension Attacks

- Concept: crypto hash function prepends key to message and then hashes as usual
- Caveat: message must not be padded
- Crypto hash is really just last chaining variable
- Therefore
  - New text appended to message
  - Original, legitimate hash set as chaining variable
  - New text and chaining variable passed into algorithm
  - Result: valid hash without knowing key

Extension Attack Example



Divide and Conquer Attack

- Strength of NMAC is measured by the length of k_1 or k_2 *(l),* not the combined length ($2l$).
- Keys are independent and independently used
  - Attacker may choose to 'parallelize' the attack, or try to break keys separately

Resolution of Attacks in NMAC/HMAC

- Birthday attacks on NMAC/HMAC only work by attacking the underlying crypto hash function
    - As an example attacker must query for 250,000 years to get even the 1.2√n messages for MD5.
- Extension Attack:
    - Does not utilize the 'prepend' construction
- Divide and Conquer
    - This is why authors assert that the strength of NMAC is specified in terms of one key *(l)* instead of the combined k_1 and k_2 (which would yield 2*l*)

Conclusions

- If NMAC/HMAC is insecure then the underlying hash is also insecure
- Efficient - minimal performance degradation caused by using a hash scheme
- If black box usage of hash functions is desired, use HMAC. Otherwise, modifying the hash function to cater to NMAC is also simple to do (in software, at least).

Questions?