

MAC's and Hash Functions

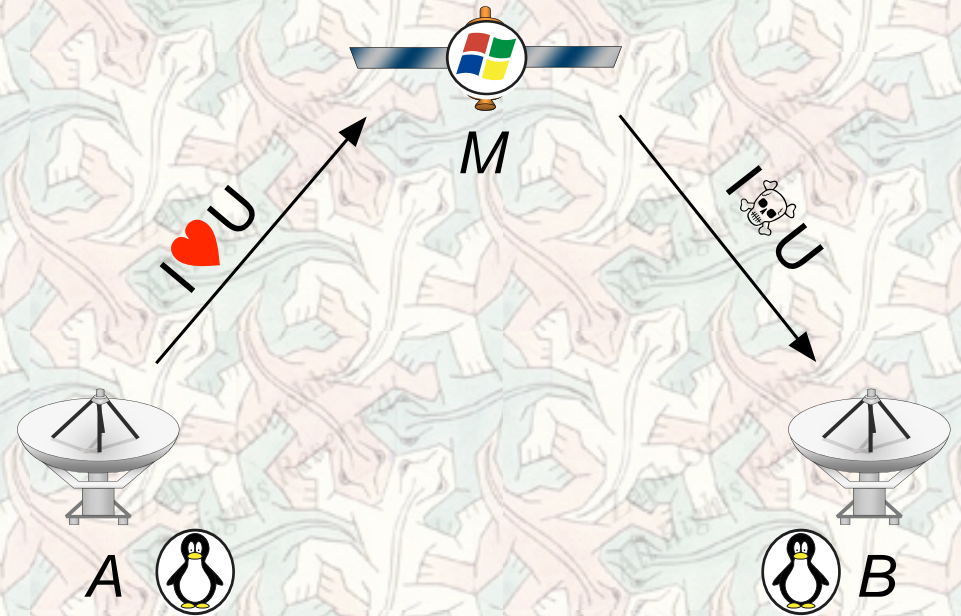
Prof. Zeph Grunschlag

Message Authentication Codes

AUTHENTICATION

PROBLEM: Alice (A) sends Bob (B) a message through insecure channel modifiable by Mallory (M)

GOAL: A encodes message in a way that enables B to detect any modification

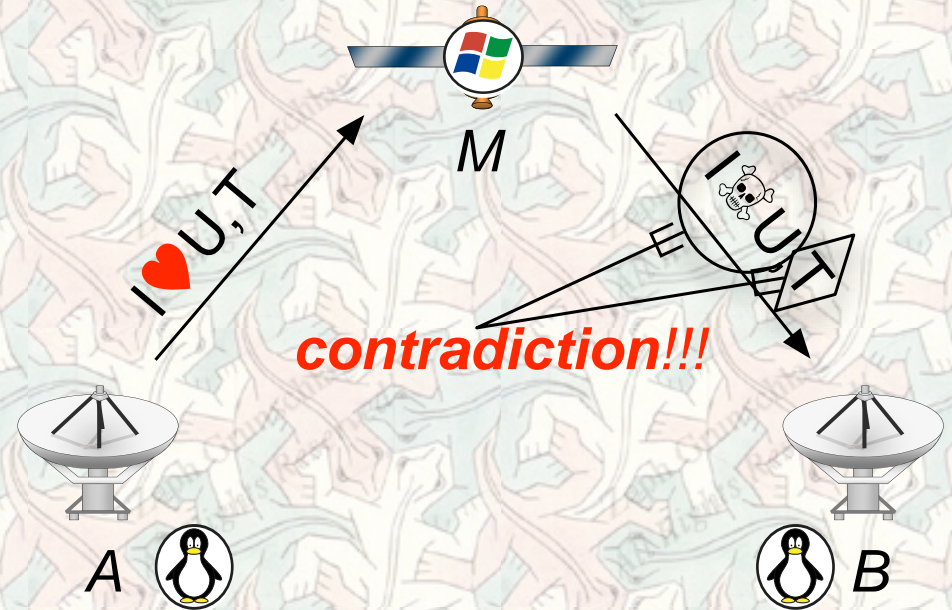


Message Authentication Codes

AUTHENTICATION

SOLUTION: A appends **MAC** tag T which changes randomly whenever message is changed

Use **hash function** to create the tag T from the message.



Desired Properties

Hash function h should satisfy:

- Output $<$ Input
- Collision resistant:

Finding different inputs with same output is computationally intractable

- One Way:

Easy to compute, hard to find pre-images

- Over insecure channel: secret keys

Hash Function Family

A HASH FAMILY is a 4-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ satisfying

1. \mathcal{X} is a (possibly infinite) set of MESSAGES
2. \mathcal{Y} is a finite set of possible TAGS (or digests)
3. \mathcal{K} is a finite set of possible KEYS
4. \mathcal{H} is a finite set of hash functions indexed by \mathcal{K} so for each $K \in \mathcal{K}$ there is a function $h_K : \mathcal{X} \rightarrow \mathcal{Y}$

Discrete Log Hash

Note: Picking specific values of message-size, tag-size, key-size. Could generalize to arbitrary but related sizes.

- 260 to 132 bit contractor:

$$\mathcal{X} = [0, 2^{260} - 1], \mathcal{Y} = [0, 2^{132} - 1]$$

- \mathcal{K} - keys $K = (p, q, \alpha, \beta)$ should satisfy:

- p and q are prime with $p = 2q + 1$

- $\alpha \neq \beta$ are **primitive** in \mathbb{Z}_p^*

- Bit-lengths: $|p| = 132, |q^2| = 261$

- \mathcal{H} - hash functions defined by

$$h_K(n) = \alpha^{\lfloor \frac{n}{q} \rfloor} \beta^{n \bmod q} \bmod p$$

Primitive Elements

DEF: An element g in a group G is said to be **primitive** (or a **generator**) if every g^i element in G can be expressed in the form for some **exponent index** i . If G contains a primitive element, G is said to be **cyclic**.

NOTE: Equivalently, g is primitive if the first positive index for which $g^i = 1$ is $i = n = |G|$.

THM: If F is a finite field, then F^* is cyclic.

COR: If p is prime, \mathbb{Z}_p^* is cyclic. Also, suppose g is primitive in \mathbb{Z}_p^* . Then g^i is primitive iff i is relatively prime to $p - 1$.

Index Calculus

Can figure out everything about how numbers *multiply* in \mathbb{Z}_p^* by seeing how their exponents (indices) *add* in $\mathbb{Z}_{\phi(p)}^+$. Generalization:

THM: If p is a prime number, then there is an isomorphism: $\mathbb{Z}_p^* \approx \mathbb{Z}_{\phi(p)}^+$.

NOTE: Isomorphism only easy to compute in (index) \rightarrow (number) direction. Other direction (number) \rightarrow (index) is DLog problem.

Discrete Logarithm Problem

DEF: Suppose that $y = x^a \pmod n$. Then a is said to be the **discrete logarithm** of y with base x modulo n . Notation: $a = \text{Dlog}_x(y) \pmod n$

Discrete logarithm assumption: No BPP algorithm $D(x,y,p)$ exists which successfully computes $\text{Dlog}_x(y) \pmod p$ with “significant” probability given a random prime p , a random primitive x in \mathbb{Z}_p^* , and a random integer y in \mathbb{Z}_p^* .

When all factors of $p-1$ are small, algorithms do exist: explains defining $(p-1)/2$ to be prime.

Computational Security of Logarithmic Hash

THM: Existence of a BPP algorithm for collision finding in discrete log hash family, implies a BPP algorithm for discrete log prob.

Note: Computational complexity definitions require considering *infinite* family of log hashes where allow arbitrarily large domains.

LEMMA: Collision resistance implies one-wayness when domain \gg codomain.

COR: Discrete log hash “is” one way.

Collisions \Rightarrow DLog

INPUT: p - prime, $x, y \in \mathbb{Z}_p^*$ with x primitive

OUTPUT: $\text{Dlog}_x(y) \bmod p$

EXTERNAL: FindCollision - assumed procedure for finding collisions in h_K

1. if $\frac{p-1}{2}$ not prime, or y not primitive “FAIL”
2. $q = \frac{p-1}{2}$, $\alpha = x$, $\beta = y$, $K = (p, q, \alpha, \beta)$
3. $(a, b) = \text{FindCollision}(K)$
4. ... continued next page ...

Collisions \Rightarrow DLog

$$i = \left\lfloor \frac{a}{q} \right\rfloor - \left\lfloor \frac{b}{q} \right\rfloor, j = (b \bmod q) - (a \bmod q)$$

$$i = i \bmod (p - 1), j = j \bmod (p - 1)$$

```
while( i mod 2 == 0 ) {
```

$$i = i / 2, j = j / 2$$

$$\text{if } ((i - j) \bmod 2 \neq 0) j = j + (p - 1) / 2$$

```
}
```

$$\text{return } [i \cdot (j^{-1} \bmod (p - 1))] \bmod (p - 1)$$

Iterated Hashes

- A procedure for repeatedly applying a particular hash function, shrinking arbitrarily long messages to fixed length tags.

EXAMPLE (SIMPLE MERKLE-DAMGÅRD):

- Assume h takes 260 bits to 132 bits and that 0^{132} is never an output. Discrete log hash (viewed on bitstring) satisfies these.
- Define buffer function b - a 1-1 function from bitstrings of length < 132 to bitstrings of length exactly 132.

SIMPLE MERKLE- DAMGÅRD

INPUT: bitstring $x = x_1x_2 \dots x_k$

OUTPUT: bitstring $y = y_1y_2 \dots y_{132}$

EXTERNAL: compression function h

//Break up into 128-bit blocks:

for $i \in [1, \lfloor \frac{k}{128} \rfloor]$ $z_i = x_{128i+1} \dots x_{128i+128}$

$z_{i+1} = b(x_{128i+1} \dots x_k)$ // buffer

$n = 0^{132}$

for each block z_i

$z = n || z_i$ // concatenate strings

$n = h(z)$ // view z as a number

return n

Security of **SIMPLE MERKLE-DAMGÅRD**

THM: Any BPP algorithm to find collisions in the **SIMPLE MERKLE-DAMGÅRD** applied to a contraction function h , would imply a BPP algorithm for finding collisions in h .

COR: If the discrete logarithm assumption holds, **SIMPLE MERKLE-DAMGÅRD** with h chosen from discrete log hash family is a secure hash family with arbitrary contraction.

NOTE: theoretically secure but impractical.