

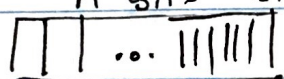
10/16

Hash Functions

def: a hash function is a computationally efficient function mapping binary strings of arbitrary lengths (no limit to input) to bin. strings of some fixed lengths, called a hash value. Output is fixed regardless of input size. e.g. whole book $\rightarrow$ 256 bits. • Various output sizes.

• Same input == same output

What is the prob. that a randomly chosen string gets mapped to a particular n -bit hash value (image).

n bits e.g. 256 bits


Answer = 2^n because
you can put a 0 or 1

a particular n would
 $b = 1/2^n$

Applications:

① Digital Signatures

- sign a hash value of a large doc. instead of the doc. itself. A long msg is usually hashed using a publicly available hash function and only the hash value is signed. Saves time + space.

② Data integrity

- assume we have msg m and its hash value h . If m is sent across an unprotected channel and manipulated, but h is sent across a secure channel, then the receiver can inject m into the same hash function and if his h equals the same h that was received, msg is ok.

hash function: Domain $\longrightarrow$ Range

$$h: D \longrightarrow R$$

$$|D| \gg |R|$$

larger

Terminology: hashcode = hash value = hash result = hash

Properties of Hash Functions

- ① Compression: h maps an input x of arbitrary/finite length to an output $h(x)$ of fixed bit lengths n .
- ② Ease of computation: Very fast. $h(x)$ is easy to compute.
- ③ Keyed hash functions and MAC, we need a 3rd property, described later.

Classes of Hash functions

- ① Modification Detection Code (MDC)
 - to provide a representative image/hash of the msg.
 - goal = data integrity
 - input = msg
- ② Msg. Authentication Codes (MAC)
 - Facilitate assurance regarding both the source of a msg, and its integrity.
 - goal = data integrity AND authentication
 - input = msg. AND a key

Security Requirements

- ① Pre-image resistance - it's comp. inf. to find any pre-image x' s.t. $h(x') = y$ when given any y for which a corresponding input is not known. Impossible to go backwards from $y \rightarrow x$ even if the public func. is known. Hence, One-way func.
- ② ^{2nd} Pre-image resistance - It's comp. inf. to find any 2nd input which has the same output as any specified input. Given x , try to find x' s.t. $x' \neq x$ but $h(x) = h(x')$. Given an input, find another input which has the same output.
- ③ Collision Resistance - it's comp. inf. to find any 2 distinct inputs x and x' which hash to the same output. $h(x) = h(x')$

Dif. between ② and ③: in ②, you are given x and asked to find x' . In ③, you can find any x and x' so they generate the same y .

Terminology: pre-image resistance $\equiv$ one way
2nd pre-image resistance $\equiv$ weak collision resistance
collision resistance $\equiv$ strong collision resistance

Summary

Unkeyed hash func: MD5

Keyed hash func: MAC

MAC def: A msg authentication code is a family of func. h_k parameterized by a secret key w . the following properties.

① ease of use/computation: $\leftarrow$ easily calculated
 $h_k, K, \underset{\substack{\uparrow \\ \text{input}}}{x}, \underbrace{h_k(x)}_{\text{MAC value, not hash value}}$

② Compression: h_k maps an input x of arbitrary finite bits to an output $h_k(x)$ of fixed bit size. 256-bit is common. SHA-256 bit used in bitcoin.

③ Computation (0:47:00) resistance: given 0 or more text/MAC pair $(x_i, h_k(x_i))$, it's comp. inf. to compute any text mac pair $(x, h_k(x))$ for any new input where $x \neq x_i$. e.g. given $(x_1, h_k(x_1)) (x_2, h_k(x_2))$ its comp. inf. to compute a new pair (x, MAC) based on this info. Main reason is becoz of the Secret Key, everything else is public. • this 3rd property is for keyed hash func.

Objectives of Adversaries vs MDC/MAC

- Adversary intends to attack MDC
 - To attack a One-Way-Func: Given a hash value y , find a pre-image x s.t. $y = h(x)$ or given a pair $(x, h(x))$, find a 2nd pre-image x' s.t. $h(x') = h(x)$
 - To attack a Collision Resistance hash func: Find any 2 inputs x, x' s.t. $h(x) = h(x')$
- Adversary intends to attack MAC
 - Weed out any prior knowledge of a key k . Compute a new text-MAC $(x, h_k(x))$ for some text $x \neq x_i$ where one or more pairs $(x_i, h_k(x_i))$ are given.

Case 1: Known text attack

- One or more text-MAC pairs $(x_i, h_k(x_i))$ are available. With these inputs, see if you can find the secret key.

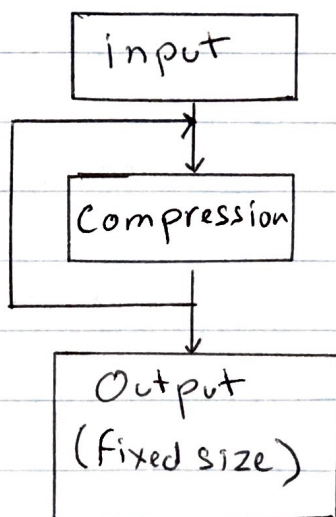
Case 2: Chosen text attack

- One or more text-MAC pairs are available. x_i is chosen by the adversary.

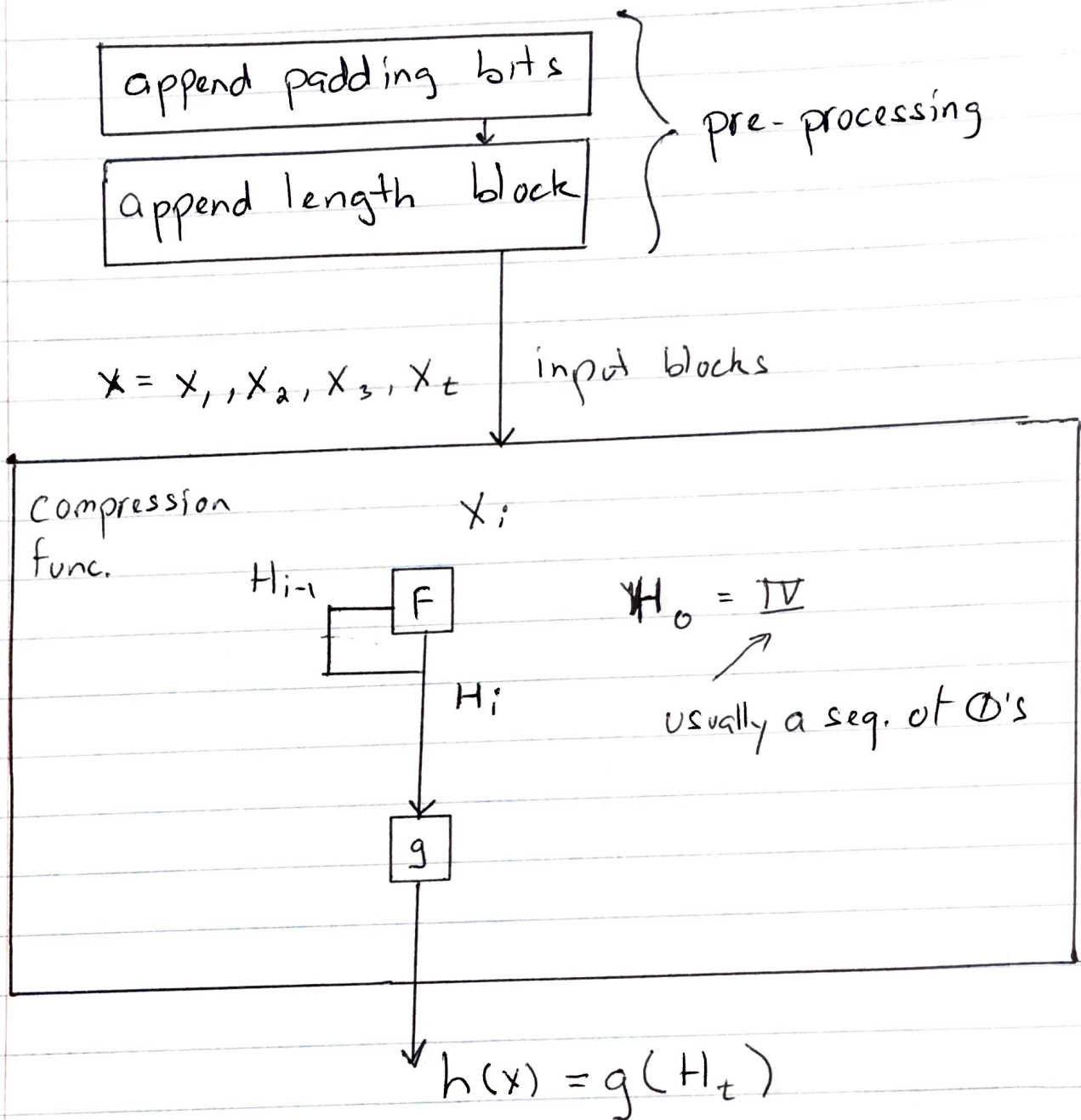
Case 3: Adaptive chosen text attack

- Most realistic
 - x_i may be chosen by the adversary as in case 2, now allowing successive choices to be based on the result of prior queries.
- e.g. Adversary selects x_i . System gives him the MAC value associated w. x_i . Does some analysis. Repeat.

Hash Func Architecture



Detail View (1:07:00)



10/18

Hash Functions based on block ciphers

① h.f. based on block ciphers

⊕ There are many efficient methods to implement these h.f. either by software or hardware. Used in digital currencies. Many are hard-wired.

② Customized h.f.

③ h.f. based on arithmetic - use math techniques to create a compression function, then do the hashing.

* Def: An (n, r) block cipher is a block cipher that defines an invertible function from n -bit plaintext to n -bit ciphertext using an r -bit key $E_k(n)$
e.g. DES(64, 56)

Def: Let h be an iterated h.f. constructed from a block cipher w. compression function f , which performs S block encs. to process each successive n -bit msg block. Then the rate is $1/S$

Single length MDCs of rate 1

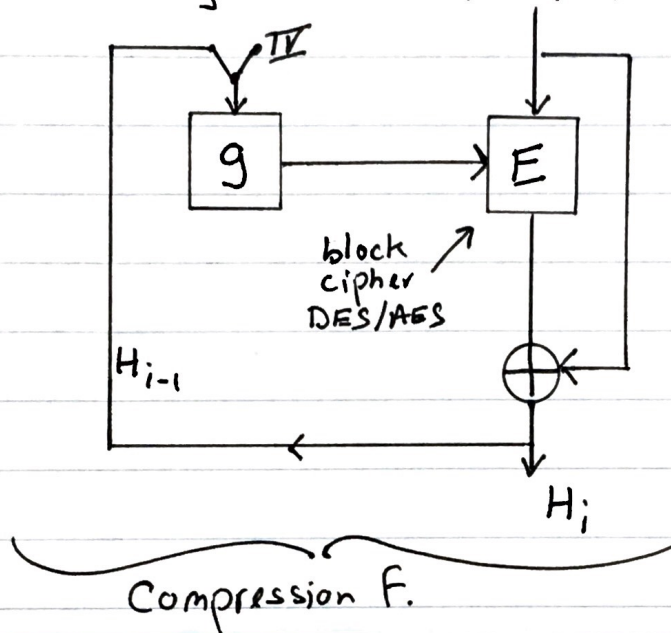
- key size is almost the same size as the block length.
- unkeyed h.f.

Requirements :

- a generic n -bit block cipher E_K .
need an enc. f. which gets 1 block of size n and does enc.
- a function g that maps n -bit inputs to keys K — we need this becoz in the MDC, it's an unkeyed h.f. so we should not rely on any secret. And if the intention is to use a generic bit cipher, or an enc. f., all these enc. f. rely on the secret, so then we need a publicly known mechanism to be able to gen. this key E_K for the enc. f. or for the block cipher. So E_K is not secret.
 - We don't have ~~to~~ secrets here, so that's why we need a way to gen. key for our block cipher.
- a fixed (usually n -bit) initial value IV.

1-1 First single length MDC name: Matyas - Meyer - Oseas

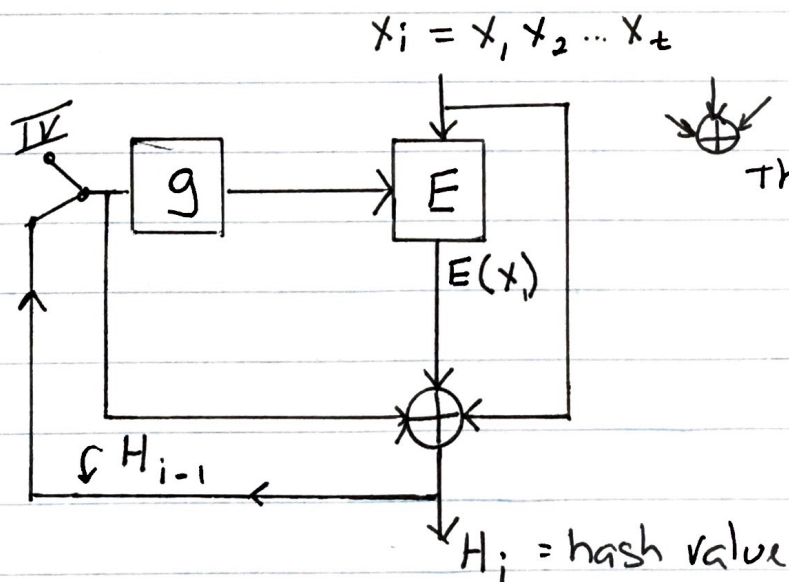
incoming blocks $x_i: x_1, x_2 \dots x_t$ t blocks of size n
input size unlimited



Last CipherText is
the block of n -bit
which will be the hash

1-3 Miyaguchi-Preneel

1-2 next page

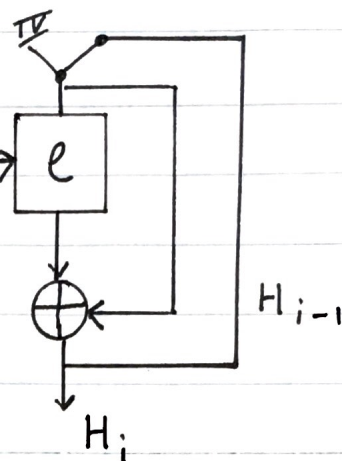


Toss all H_i , keep the last one

1-2 Davis - Meyer

input $X_i: x_1, x_2 \dots x_t$

These will be used as the key.



Formal Version

1-1 mmo

input: bit string $x \xrightarrow{x_0} x_1, x_2 \dots x_t$

output: n-bit hash code/value of x

① x is divided into n-bit blocks and padded w. zeroes. Define IV

② $H_t: H_0 = \text{IV}$

$H_i = E_{g(H_{i-1})}(x_i) \oplus x_i$ } unkeyed h.f./MDC

1-2 DM

$$H_t = H_0 = \text{IV}$$

$$H_i = E_{x_i}(H_{i-1}) \oplus H_{i-1}$$

1-3

$$H_t = H_0 = \text{IV}$$

$$H_i = (E_{g(H_{i-1})}(x_i) \oplus H_{i-1}) \oplus x_i$$

Double length MDC - unkeyed h.f.

Requirements:

(a) DES (not 3DES) used as a compressor.

bit lengths = 64 key size = 56

(b) g and $\tilde{g}$ functions that map 64 bit input $\rightarrow$ 56 bit output.

$$g(u) = u_1 \ 1 \ 0 \ u_4 \dots u_7 \ X \ u_9 \dots u_{63}$$

- exclude $u_8 \ u_{16} \ u_{24} \dots u_{64}$

- replace bits 2 and 3 w. 1 0

0:45

$$\tilde{g}(u) = \text{Convert 64-bit input to } \rightarrow$$

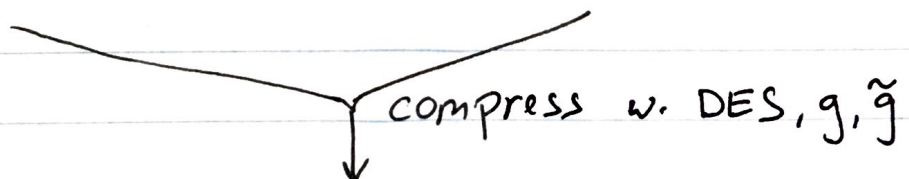
$$= u_1 \ 0 \ 1 \ u_4 \dots u_7 \ u_9 \dots u_{63}$$

- remove same 8 bits

- replace bits 2 and 3 w. 0 1

- chunk large input file into 64-bit blocks
- pad last block w. 0's
- Output is 128 bits

... | 64 bit | 64 bit | ... | 64 bit | ...

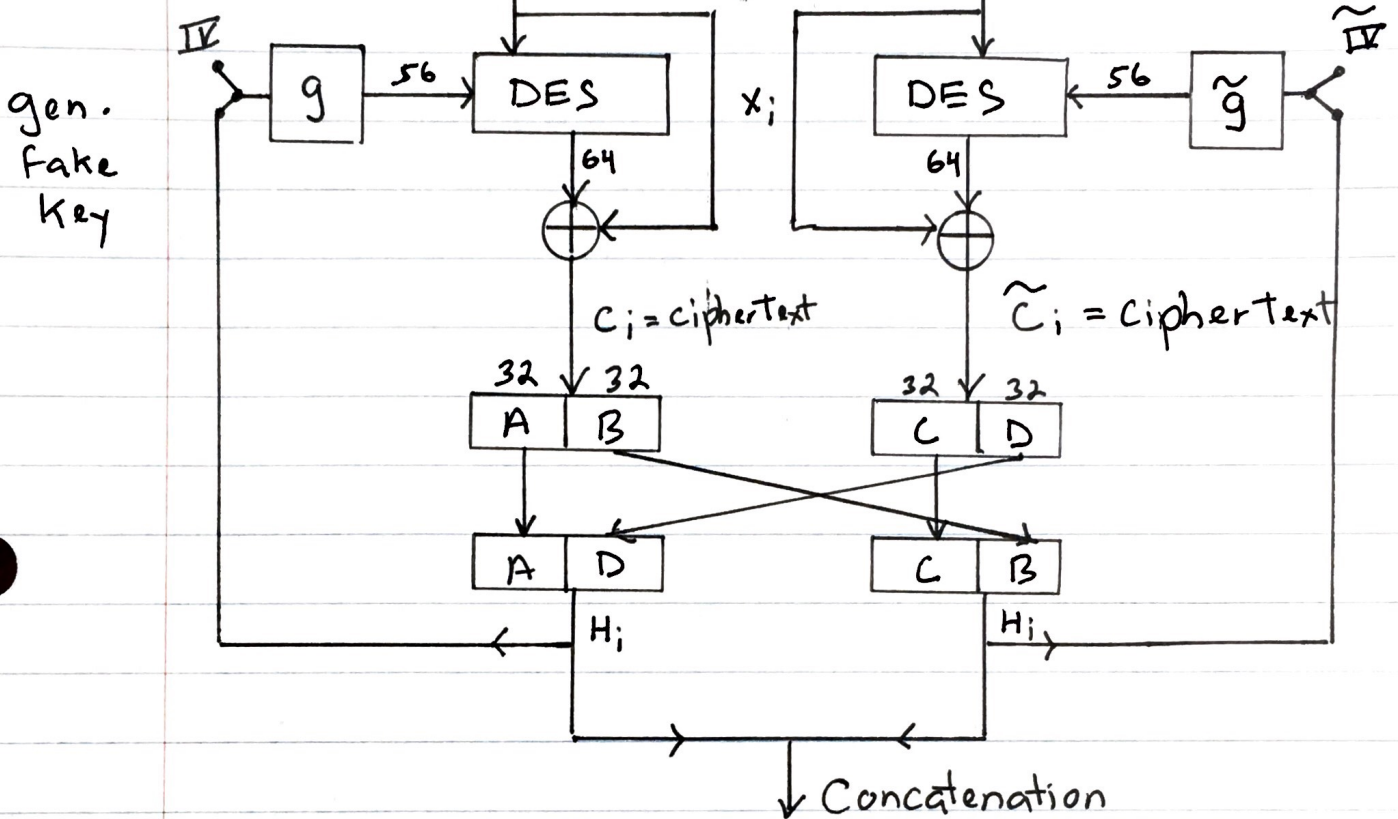


128 bit hash value.

• No secrets, this is unkeyed h.f.

Architecture

$X_i = X_1, X_2 \dots X_t$ 64-bit blocks, 1st one is padded



$$H = H_t \parallel \tilde{H}_t$$

128-bit h.v.

$$H_0 = IV$$

$$\tilde{H}_0 = \tilde{IV}$$

$$\left. \begin{aligned} C_i &= E_g(H_{i-1})(x_i) \oplus x_i \\ \tilde{C}_i &= E_{\tilde{g}}(\tilde{H}_{i-1})(x_i) \oplus x_i \end{aligned} \right\} \begin{aligned} H_i &= C_i^L \parallel \tilde{C}_i^R \\ \tilde{H}_i &= \tilde{C}_i^L \parallel C_i^R \end{aligned}$$

10/23

MAC based on block ciphers - uses secret key
CBC-MAC Alg.

Input: data x , block cipher E (symmetric enc. scheme), Secret key for E .

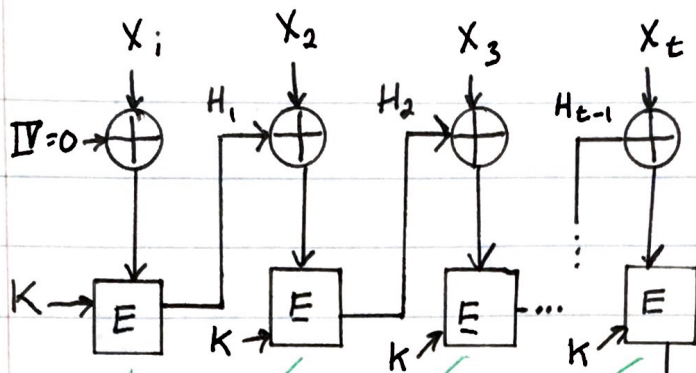
Output: n -bit MAC on x (n is the block size for E)

- ① Padding and blocking (creating blocks of input)
- Pad x if necessary, then divide the padded text into n -bit blocks denoted by $x_1, \dots, x_t$
- ② Processing: Compute h_t :

$$H_1 \leftarrow E_K(x_1) \quad IV = 0 \parallel E_K(IV \oplus x_1)$$
$$H_i \leftarrow E_K(H_{i-1} \oplus x_i) \quad | \text{ for } 2 \leq i \leq t$$

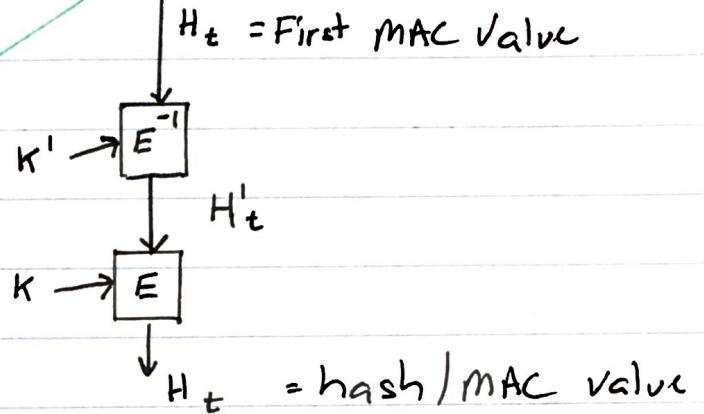
- ③ Optional: process to increase the strength of MAC. Choose a 2nd secret key K' $K' \neq K$. Then $H'_t \leftarrow E_{K'}^{-1}(H_t)$ - decrypt using new secret key the last hash value.
 $H_t \leftarrow E_K(H'_t)$

- ④ n -bit block H_t is going to be our hash/MAC value. and we can use this MAC value for integrity and auth.



Architecture.

Compression, not enc.



Hash Functions based on modular arithmetic

MASH (1995): Modular arithmetic secure h.f.

Input: data x of bit length $0 \leq b \leq 2^{n/2}$

Output: n -bit hash of x .

n is almost the bit length of our modulus.

if $n = 48$ bits (output size) then input can be $2^{48/2} = 2^{24}$ (very large input).

$$2^5 < 48 < 2^6 \rightarrow 32 < 48 < 64$$

① Setup: Select 2 primes of bit length m .

$m = pq$ (assume factorization is comp. inf.). Then the bit length of n of the hash result should be the largest multiple of 16 less than m .

Assume $m = 64$ bits

$$n = 16 \cdot n' < m$$

n is a multiple of 16

$$n = \begin{matrix} & 16 & 32 & 48 & 64 \\ & & & n & m \end{matrix}$$

so whatever is n , go to next multiple of 16 and that will be n .

$H_0 = 0$ as IV

$A = 0 \times f 0 0 \dots 0$ -constant / public

V = inclusive or

$n / 2^{n/2}$ are known ahead of time so m can be calculated as well. m defines the size of the modulus.

② Setup padding + blocking

- pad x w. 0 bits if necessary to obtain $t \cdot n/2$ - bit input. Each block will be $n/2$ bit.

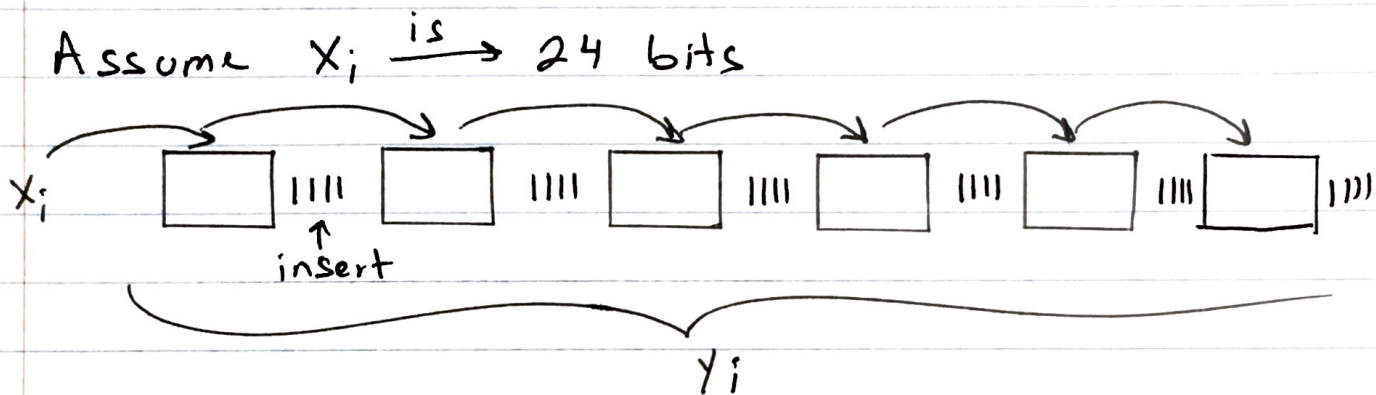
So if you selected the hash value to be 48 bits, then each block will be 24 bits. And we have t of these blocks.

Then divide x (the entire input) into t blocks of size $n/2$.

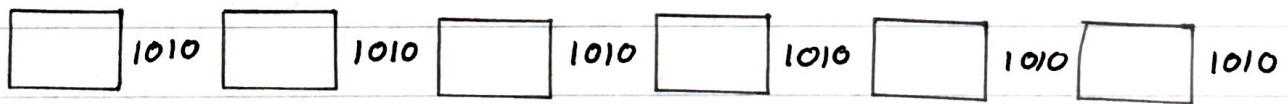
Finally, X_{t+1} denotes $n/2$ bit representation of b bit length of input x $0 \leq b \leq 2^{n/2}$

X_{t+1} is an extra block - this last block is the $n/2$ bit rep. of b . which is the actual size of the input.

- ③ Expand each block x_i ($n/2$ bit = 24 in our e.g.) to an n -bit block y_i by partitioning it into 4-bit nibbles and inserting 4 1-bits preceding each, except for y_{t+1} in which the inserted nibble is 1010 (not 1111). He said 0101 but wrote 1010. The intention of this stage is to double to 48 bits.



$x_{t+1} \xrightarrow{\text{to}} y_{t+1}$ do the same



?

0:55

④ Compression function $1 \leq i \leq t+1$

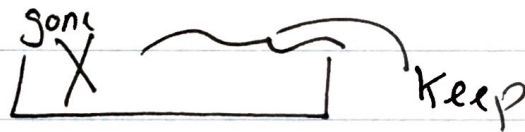
repeat this for $i=1$ to $t+1$

map 2 n -bit inputs h_{i-1} and y_i to one n -bit to gen. h_i :

$$h_i \leftarrow (((h_{i-1} \oplus y_i) \vee A)^2 \bmod M \rightarrow n) \oplus h_{i-1}$$

$\uparrow$ $H_0 = IV = 0$ $\uparrow$ constant, see wiki

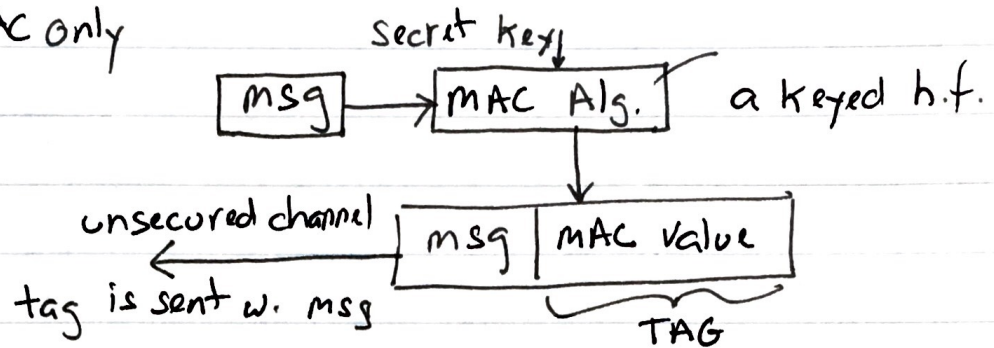
$\rightarrow$ denotes keeping the rightmost n -bits of the m -bit result.



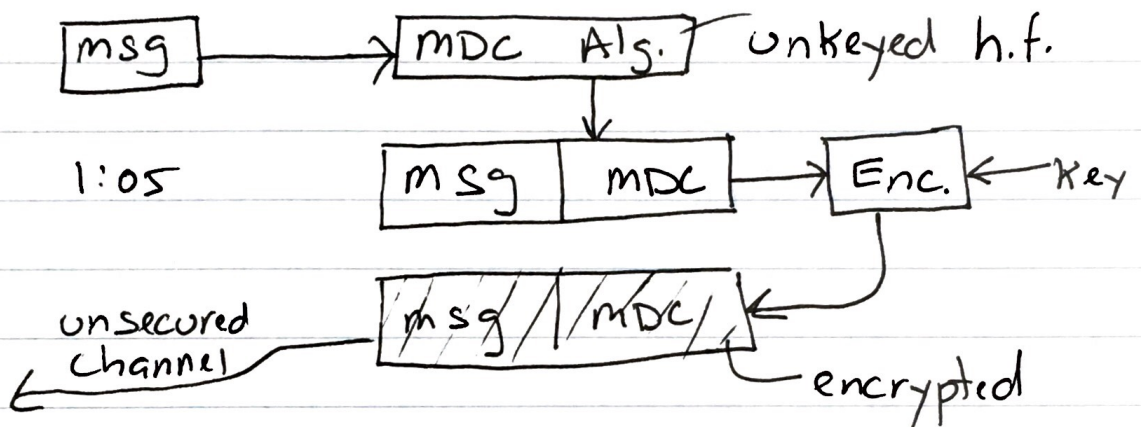
m defines the size of M .

3 Methods for Data Integrity using h.f.

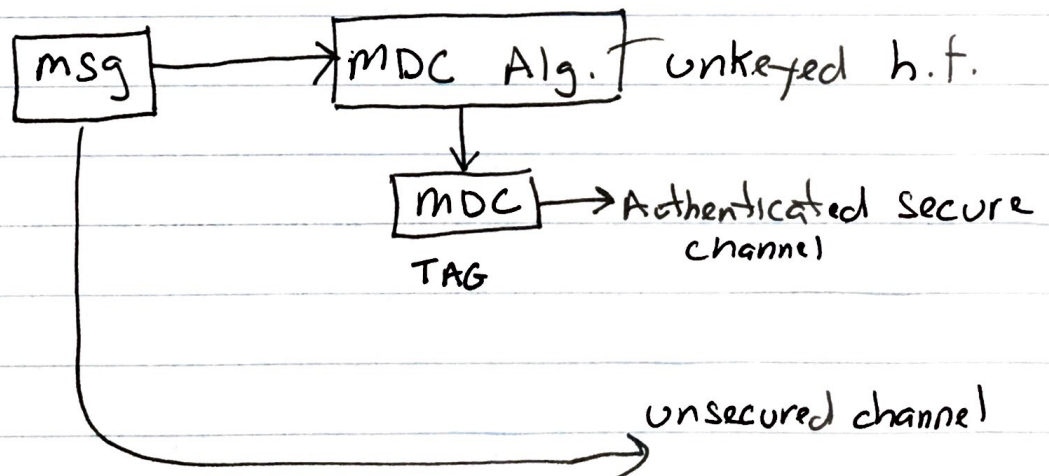
(a) MAC only



(b)



(c) MDC and authenticated channel



10/25

Blockchain & Bitcoin

any input size $\rightarrow$ h.f. $\rightarrow$ 4 bits $\rightarrow 2^4 = 16$ possible values

now make a h.f. where the first bit is 0

$\rightarrow$ h.f. $\rightarrow$

0	1	1	1
---	---	---	---

of the 16 possibilities, 8 satisfy the condition.

Terminology

- Transactions are grouped in the bitcoin network in blocks in order to be verified by a subset of nodes in the BC network known as miners.
- The mining process, aka proof-of-work is computationally intensive (not comp. inf.) w. a difficulty factor that is increased over time as the computational power of hardware systems or miners grows.
- Nodes for mining pool under the supervision of pool managers to accomplish the mining task.
- The first mining pool that accomplishes the proof-of-work is rewarded w. BC

- As soon as a block is verified, it is attached to the list of existing verified blocks, aka block chain. Immediately after that, miners stop the mining process of the verified block and start working on the next one.
- The hashing rate / mining power is the total of h.v. that a minor can calc. during a specific time interval. The pool mgr. distributes the revenue among miners based on their mining powers.

block format

block #	77	
Nonce	27123	
Transactions	25B from X → Y	
hash value of prev. block		
Current:	This is what we calc	

Randomly change
2000 Transactions

Take this block and randomly change nonce, then gen. a h.v. that satisfies some condition.

Dealing w a large h.f. like 256, the total no. of possibilities are 2^{256} . If the criteria is that the result needs 2 leading 0's, then you have to calc. how many of the 2^{256} start w. 20 0's. $\frac{\text{possibilities}}{2^{256}}$

- 10 min is a fixed value to solve.

Dishonest Mining Strategies

- ① Block withholding attack - a dishonest miner only reveals a partial solution of the verification problem where he has the complete solution in order to act in favor of another competing coalition.
- ② Selfish mining - the miners of a coalition keep their discovered blocks private and continue to verify more blocks privately until they get a subchain that is larger than verified blocks.
- ③ Eclipse attack - makes a node invisible in the network, i.e. a single node monopolize all possible connections to a victim and eclipses from the network.
- ④ Stubborn mining - mining on its private chain more than the selfish mining strategy.

11/6

Digital Signatures (DS)

DS from reversible public enc. scheme.

PKE - Reversible - Suppose E_e is a public key enc. scheme w. msg spc. M and cipher spc C , let $M=C$. e.g. you can use the alphabet for M and C only. Suppose D_d (the dec. f.) where d is the private key. If we have the following condition, then we have a reversible scheme.

$$D_d(E_e(m)) = E_e(D_d(m)) = m$$

for all $m \in M$

private key public key

* not true for all enc. schemes.

Construction of a simple DS scheme

- ① Let M be the msg spc.
- ② Let $M = C$ where $C = \text{sig. spc.}$
- ③ Let (e, d) be a pair of keys of the scheme.
- ④ Signing f. $S_A : S = D_d(m)$

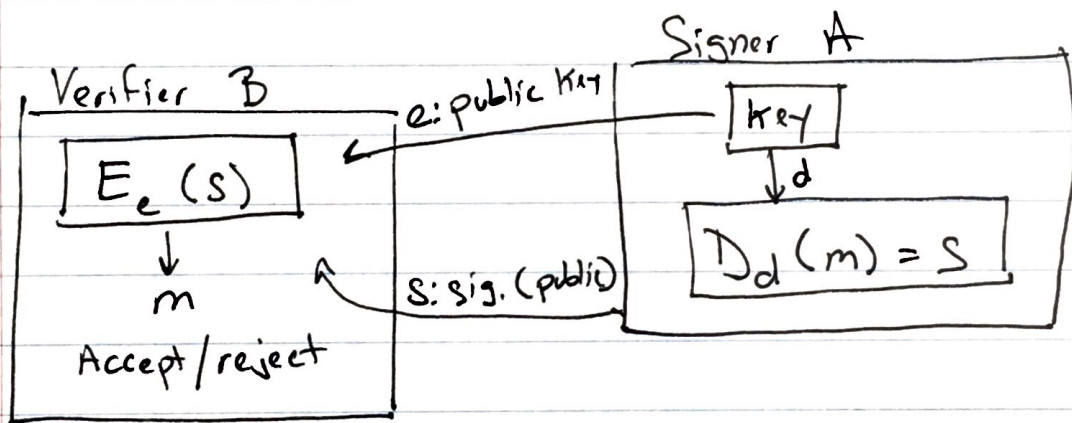
↑
defines

↑
sig.

↑
Private key

↑
msg

- ⑤ Verification f. $V_A = V_A(m, s) = \text{true} / \text{false}$
true if $E_e(s) = m \in M$
false otherwise



Requirements (of a DS scheme)

- easy to compute by signer
- easy to verify
- be computationally secure from forgery

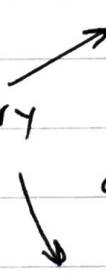
Terminology

- Resolution of Disputes

e.g. 'A' could at some point deny having signed a msg. or the other entity 'B' could falsely claim that a sig. on a msg. was produced by 'A'.

Basic definitions

- a DS is a data string which associates a msg (in digital format) w. someone originating entity(s).
 - DS gen. alg.
 - DS verification alg.
- } this combo = a DS scheme

DS schemes: category 

DS schemes w. appendix

require the original msg as input to the verification alg.

DS scheme w. msg. recovery

don't require the original msg as input to the verification alg. The original msg. is recovered from the signature.

Terminology

- ① msg 'M' is the set of el. to which a signer can affix a DS.
- ② signing spec. M_s is the set of el. to which the sig. transformations are applied.
Note: the sig. transformations are not applied directly to el. of M.
- ③ Sig. spec. S is the set of el. associated w. msg in M. Candidate sig. are going from S. Purpose: to bind the signer to a msg.
- ④ Indexing set R is used to identify a specific signing transformation.

DS def: A DS scheme (appendix or msg recovery) is said to be a randomized DS scheme if the cardinality $R > 1$ $|R| > 1$

This means more than 1 signing transformation is available. Don't necessarily need to use the same exact function for signing.

Otherwise the DS scheme is said to be deterministic, aka one-time-signing schemes.

Alg. for Sig. gen. + Verification

① Sig. gen. A (Alice)

①.1 Select an el. $K \in R$ (the transformation spec.)

ie. if you have 3 dif. transformation f .

then K can have a value $1..3$ and

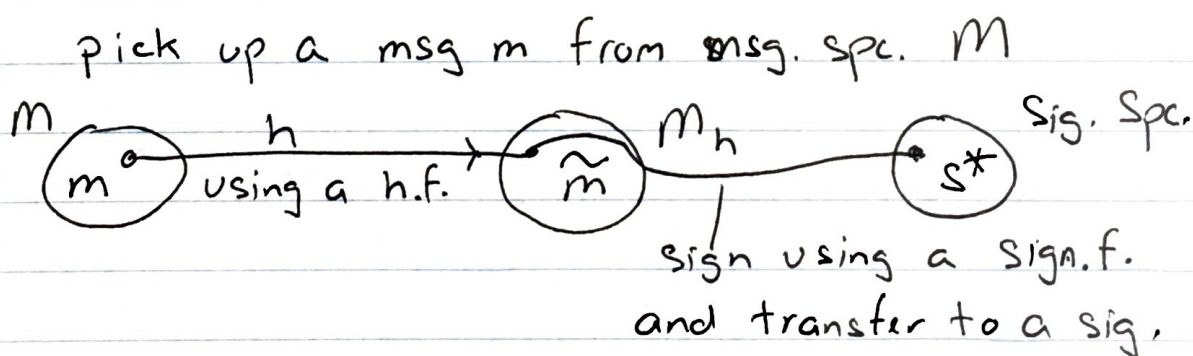
based on that, you will select your transformation f . $K \in R$ defines which trans. f should be used.

①.2 Compute the h.v. of msg. $\tilde{m} = h(m)$

then apply sign.f. $S_{A,K}(\tilde{m}) = S^*$

sig. f. gen. sig.

①.3 both m and S^* are made available to B for the verification.



- sign h.v., not the original msg.

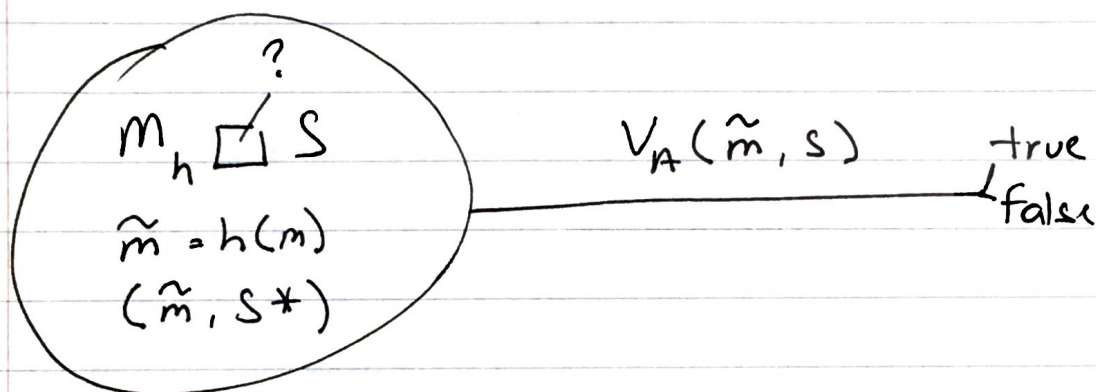
② Verification - done by other party B.

②.1 Obtain A's authentic public key

②.2 compute $\tilde{m} = h(m)$ and calc.

$$u = V_A(\tilde{m}, s^*)$$

②.3 accept the sig. iff $u = \text{true}$



- For DS schemes w. msg. recovery, we can use RSA and Rabin (next class) won't need msg. w. sig. it can be extracted from sig.

11/8

DS scheme w msg. recovery

Def: is a scheme for which prior knowledge of the msg is not required for the verification alg. $\rightarrow$ RSA, Rabin

Alg. for key gen.

Summary: Create a private key to be used by signer and public key to be used by verifier.

① 'A' should select a set $SA = \{S_{A,k} : k \in R\}$ of transformations. Each $S_{A,k}$ is a one-to-one mapping from $\underbrace{M_s}_{\text{signing spc.}} \rightarrow S$

msg. spc.

Signing spc.

m

m_s

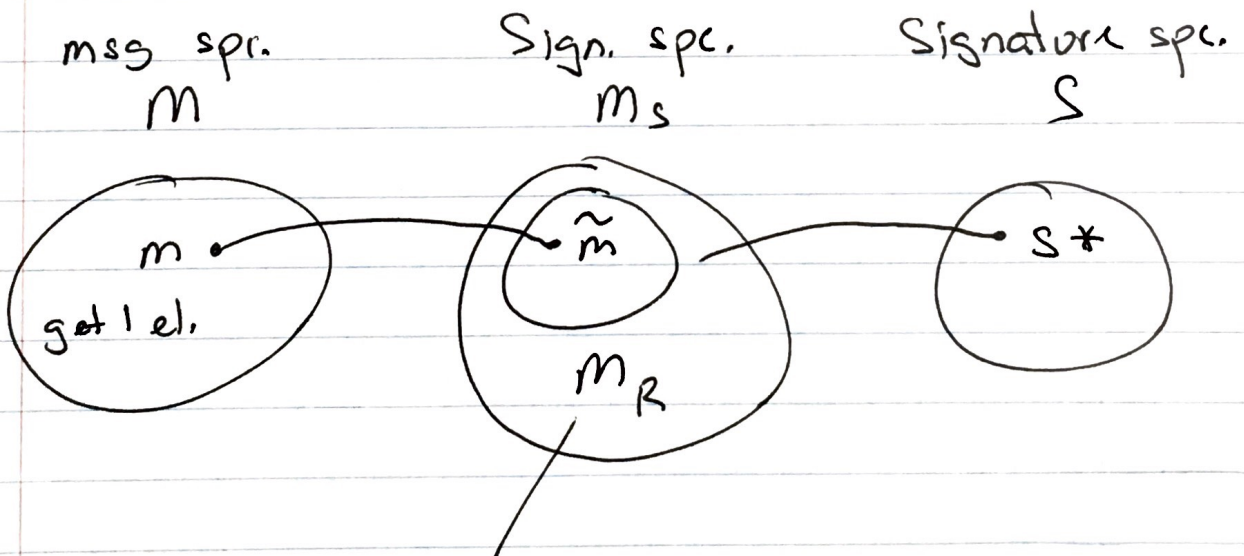
S

$\underbrace{\hspace{1cm}}$
h.v. of
all possible mss.

Signing
Sig. spc.

② S_A defines a corresponding mapping V_A w. the property that $V_A \circ S_{A,k}$ is the identity map on M_S for all $k \in R$. V_A : verification transformation.
In the prev. case, it took a msg and a sig. and transformed it to true or false. But here, we don't need the msg.

③ We need a public key for V_A (the verification alg.) and a private key for the sig./transformation S_A



another set M_R - a subset of M_S . $M_R \subseteq M_S$

Algs. for sign. + ver. process

① Sig. gen. by 'A'

(1.1) select an el. $k \in \mathcal{K} \rightarrow$ to select the transformation f .

(1.2) Compute $\tilde{m} = R(m)$ $\rightarrow$ not the same
 $\quad \quad \quad \hookrightarrow$ a redundancy f.

comp. sig. $S^* = S_{A,k}(\tilde{m})$

↳ by using the sign. trans.

1.3 sig. S^* is sent to 'B'. 'B' verifies the sig. and recovers the msg. from S^* . Only sig. is sent to B, not the msg.

② Verification: $\exists$

2.1 Obtain 'A's authentic public key for verification alg. V_A .

(2.2) Comp. $\tilde{m} = V_A (S^+)$

(2.3) Verify that $\tilde{m} \in M_R$. If false, reject sig.

(2.4) Recover msg. from $\tilde{m}$ by using the inverse of R (the redundancy func)

So R goes from msg. spec. to m_R

$$R: M \rightarrow m_R$$

subst of m_s

then

S goes from sign. spec. m_s to sig. spec. S .

$$S: m_s \rightarrow S$$

then

R^{-1} goes from m_R to m

$$R^{-1}: m_R \rightarrow m$$

Requirements

- ① for each k in the sign. f. $S_{A,k}$ should be efficient to comp.
- ② The verification f. V_A should be efficient to comp.
- ③ Security req: It should be comp. inf. for an entity other than 'A' to find any sig. in the sig. spc. $s^* \in S$ s.t.
 $V_A(s^*) \in M_R \rightarrow$ generating a valid sig. (forging a sig.)
be a valid sig.

Imp w/ RSA

RSA sig. scheme

$$\mathbb{Z}_n = \{0, 1, \dots, n-1\} \quad n = pq$$

sign. + sig. spc.

sign spc M_S and sig spc S^* are

$R: M \xrightarrow{to} \mathbb{Z}_n$ R is a f. from msg spc.

Alg. for key gen.

1. select 2 large primes p, q
2. Calc n . and $\phi = (p-1)(q-1)$
3. a rand. no. e will be selected s.t.
 $\gcd(e, \phi) = 1$ aka relatively prime
4. Use EE Alg to comp. d $1 < d < \phi$ s.t.
 $ed \equiv 1 \pmod{\phi}$. e and d are M of each other.
5. (n, e) are public, d is private
use d to sign
use n, e to verify
reverse order of RSA

2 RSA sig. gen. + ver.

1. sig. gen
 - 1.1 pick up your redundancy f. (wiki, etc)
 $\tilde{m} = R(m)$ an int $[0, n-1]$
 - 1.2 gen $S = \tilde{m}^d \pmod{n}$
 - 1.3 sig for m is S . S is sent to other guy.
2. Ver. S is valid and gen m from S .
 - 2.1 obtain authentic public key n, e
 - 2.2 comp $\tilde{m} = S^e \pmod{n}$
 - 2.3 ver $\tilde{m} \in M_R$ if not, reject
 - 2.4 recover $m = R^{-1}(\tilde{m})$

↑ not a one-way f, but it compresses msg

e.g. RSA DS scheme w msg rec.

$$p = 7927 \quad q = 6997 \quad n = pq = 55,465,219$$

$$d = 7926 \cdot 6996 = 55,450,296$$

$$e = 5 \longrightarrow se \equiv 1 \pmod{55,450,296}$$

$$\begin{array}{c} | \\ \text{EEA's} \end{array} \longrightarrow d = 44,360,237$$

$$m = \mathbb{Z}_{55,465,219} \quad // \text{ msg spec.}$$

Redundancy f. R goes from m to $\mathbb{Z}_n$

aka $R: m \rightarrow \mathbb{Z}_n$ is the identity map (everything is mapped to itself, no compression) $R(m) = m$
for all $m \in m$

$\hookrightarrow$ msg spec.

$$m = 31229978 \in \mathbb{Z}_n \longrightarrow \tilde{m} = 31 \text{ same} \text{ still in } \mathbb{Z}_n$$

$$\text{Sig. } S = \tilde{m}^d \pmod{n} = 30729435$$

$$\text{ver. } \tilde{m} = S^e \pmod{n} = 31229978 \in \mathbb{Z}_n \text{ valid}$$

$$m = R^{-1}(\tilde{m}) = 31229978$$