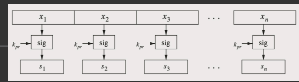
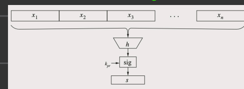
Hash Functions:

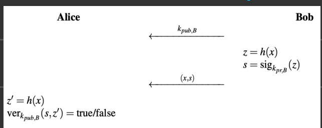
→ Hash Functions are good for the use of digital signatures → How can we efficiently compute signatures of large messages? → Divide message x into blocks x_i with size ≤ allowable length & sign each block separately



→ Problems w/ Approach above:
1. High Computational Load: Intensive asymmetric operations like modular exponentiation → signer & verifier spend large amt. of time
2. Bandwidth Overhead: Many signatures + Large Storage ⇒ Overhead
3. Security Limitations: Block Based Signatures ⇒ Attacks by manipulating each block (need one short signature) → Sol^{n}: Hash F^{n} that computes finger print of message x



→ Basic Protocol for Digital Signature with Hash Functions: Bob wants to send digitally signed message to Alice



Computes hash of x, z, & signs z with his key $k_{pr,B}$.

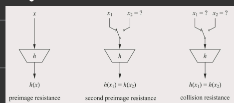
Alice computes hash z' of received x. She verifies signature s w/ Bob's public key $k_{pub,B}$

Signature generation & verification work on z ⇒ Hash = message digest → Want hash f^{n} h to be computationally efficient for any x. Output of h is fixed length, independent of $|x|$

∴ $h: \{0,1\}^* \to \{0,1\}^n$ → Modern: $256 \le n \le 512$ bits

Computed footprint sensitive to all input bits

→ For Hash F^{n} to be secure they must possess:
1. Preimage Resistance
2. 2nd Preimage Resistance / Weak Collision Resistance
3. Collision Resistance



Say Bob encrypts message but not signature: $(e_k(x), \mathrm{sig}_{k_{pr,B}}(z))$ (RSA; symmetric)

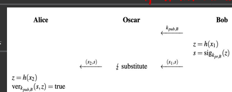
$s = \mathrm{sig}_{k_{pr,B}}(z) \equiv z^d \bmod n$ — Oscar uses Bob's public key e to compute $s^e \equiv z \bmod n$

If h not one-way Oscar can compute x from $h^{-1}(z) = x$

→ Preimage Resistance: Hash f^{n} must be one-way. Given output z, must be computationally infeasible to derive x s.t. $h(x) = z$

→ Weak Collision Resistance: Must be computationally infeasible for Oscar to create 2 diff $x_1 \ne x_2$ s.t. $h(x_1) = h(x_2)$

↳ Case 1: x_1 given ⇒ Find x_2 ↳ Case 2: Oscar free to choose both x_1 & x_2



→ Ex: Bob hashes & signs x_1. If Oscar can find x_2 s.t. $h(x_1) = h(x_2)$ he can run Substitution Attack.

Alice accepts x_2 as message since verification true since ver is based on signature

How can we prevent Oscar from finding x_2?: Use Hash F^{n} w/ no Weak Collisions → impossible due to Pigeonhole Principle

Sol^{n}: Strong Hash F^{n} designed s.t. Given x_1 & $h(x_1)$ impossible for $x_2 \ne x_1$ s.t. $h(x_1) = h(x_2)$ → no analytical attack; → BFA is possible

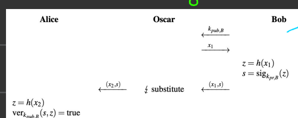
→ Collision Resistance: Computationally infeasible to derive $x_1 \ne x_2$ s.t. $h(x_1) = h(x_2)$

Ex Attack:

x_1 = Transfer \$10 into Oscar's account
x_2 = Transfer \$10,000 into Oscar's account

Oscar alters x_1 & x_2 at nonvisible locations until $h(x_1) = h(x_2)$

Yuval's Birthday Attack → If attacker has say 64 alterable locations ⇒ yield 2^{64} versions w/ 2^{64} diff hash values → Launch following attack.



Can trick Bob into signing x_1

First guess: If hash f^{n} has output length of 80 bits check 2^{80} messages; attacker only needs 2^{40} messages based on birthday paradox

How many people required at a party s.t. reasonable chance that at least 2 people have the same birthday.

Piecewise Approach: First find probability of 2 people not having same birthday

$P(\text{no collision among } 2) = 1 - \frac{1}{365}$

$P(\text{no collision among } t) = \left(1 - \frac{1}{365}\right) \cdots \left(1 - \frac{t-1}{365}\right)$

$P(\text{at least 1 collision}) = 1 - P(\text{no collision}) \approx 50\%$ for 23; $\approx 90\%$ for 40

→ Collision Search for Hash F^{n}: 2^n values where n = output width of h

How many messages $(x_1, \ldots, x_t)$ Oscar needs to hash to have reasonable chance of $h(x_i) = h(x_j)$ for $x_i \ne x_j$

$$P(\text{no collision}) = \left(1 - \frac{1}{2^n}\right) \cdots \left(1 - \frac{t-1}{2^n}\right)$$

$$= \prod_{i=1}^{t-1}\left(1 - \frac{i}{2^n}\right) \approx \prod_{i=1}^{t-1} e^{-\frac{i}{2^n}} \approx e^{-\frac{1+2+3+\cdots+t-1}{2^n}}$$

$$\approx e^{-\frac{t(t-1)}{2^{n+1}}}$$

$\lambda = 1 - P(\text{no collision})$

$$t \approx 2^{\frac{n+1}{2}} \sqrt{\ln\left(\frac{1}{1-\lambda}\right)}$$

$$\approx 2^{\frac{n}{2}} = \sqrt{2^n}$$

λ	Hash output length [bits] 128	160	256	384	512
0.5	2^{64}	2^{81}	2^{129}	2^{193}	2^{257}
0.9	2^{65}	2^{82}	2^{130}	2^{194}	2^{258}

Properties of Hash Functions
1. **Arbitrary message size** $h(x)$ can be applied to messages x of any size.
2. **Fixed output length** $h(x)$ produces a hash value z of fixed length.
3. **Efficiency** $h(x)$ is relatively easy to compute.
4. **Preimage resistance** For a given output z, it is computationally infeasible to find any input x such that $h(x) = z$, i.e, $h(x)$ is one-way.
5. **Second preimage resistance** Given x_1, and thus $h(x_1)$, it is computationally infeasible to find any $x_2 \ne x_1$ such that $h(x_1) = h(x_2)$.
6. **Collision resistance** It is computationally infeasible to find any pair $x_1 \ne x_2$ such that $h(x_1) = h(x_2)$.

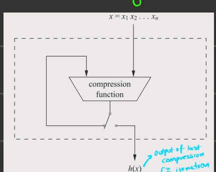
→ Hash Functions
- → Dedicated: Designed to serve as hash f^{n} (More popular)
- → Block Cipher Based: Possible to use block ciphers like AES to make Hash F^{n}

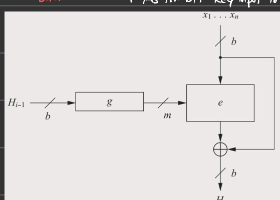
→ Hash F^{n} process arbitrary length message → Fixed Length Output (Segmenting Input = Equal size blocks processed sequentially by hash)

Ex: Merkle-Damgard Construction (Basis for SHA-1 & SHA-2)

→ Hash F^{n} from Block Ciphers: Divide x into blocks x_i of fixed size which are encrypted w/ block cipher e of size b

↳ As m-bit key input to cipher, use H_{i-1}, previous output, processed through g. $b \to m$ bit mapping. $H_n = h(x) = h(x_1, \ldots, x_n)$

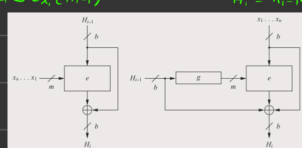




$H_i = e_{g(H_{i-1})}(x_i) \oplus x_i$ ← Matyas-Meyer-Oseas Hash F^{n}

Davies-Meyer: $H_i = H_{i-1} \oplus e_{x_i}(H_{i-1})$

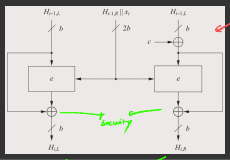
Miyaguchi Preneel: $H_i = H_{i-1} \oplus x_i \oplus e_{g(H_{i-1})}(x_i)$



→ All Hash F^{n} need initial values for H_0 → For Preimage & Weak Collision Resistance: AES-128

→ For Collision Resistance: Rijndael, block cipher that became AES w/ block width of 192 or 256 bits

another way of getting larger digests is use constructions that are composed of several instances of block cipher → yield twice width of b



Hirose Construction:
cipher e is employed
k length = 2 block length
AES-256 ex

→ Most Popular Dedicated Hash F^{Σ}: SHA-1, SHA-2, SHA-3

Algorithm		Output [bits]	Input [bits]	# rounds	Collisions found
MD5		128	512	64	yes
SHA-1		160	512	80	yes
SHA-2	SHA-224	224	512	64	no
	SHA-256	256	512	64	no
	SHA-384	384	1024	80	no
	SHA-512	512	1024	80	no

Compression f^{Σ} works like a block cipher Feistel

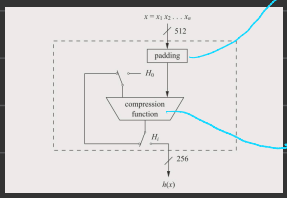
→ Most popular message digest f^{Σ} based on Merkle–Damgard

SHA-256 produces 256-bit outputs from max input length of $2^{64}-1$ bits

↓

compression f^{Σ} processes message in 512-bit chunks w/ 64 rounds

$(H_{0,L} \| H_{0,R})$

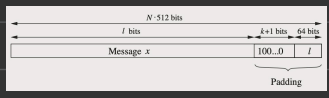


→ Before computation, X padded to multiple of 512 bits

Assume X has length l

$k = 448 - (l+1) \bmod 512$

To obtain multiple of 512 bits length, append single "1" followed by k zero bits and a 64-bit representation of l



Before applying compression f^{Σ}, divide message into 512 bit block $X_1, \ldots, X_m$

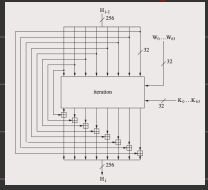
where each $X_i = (X_i^{(0)} X_i^{(1)} \cdots X_i^{(k)})$ where $X_i^{(k)}$ are 32 bit words

→ 3 inputs: H_{i-1}, Message Schedule = $W_0, W_1, \ldots, W_{63}$, constants $K_0, \ldots, K_{63}$

$$W_0 = \begin{cases} X_i^{(j)} & 0 \le j \le 15 \\ \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16} & 16 \le j \le 63 \end{cases}$$

$\sigma_0(x) = ROTR^7(x) \oplus ROTR^{18}(x) \oplus SHR^3(x)$

$\sigma_1(x) = ROTR^{17}(x) \oplus ROTR^{19}(x) \oplus SHR^{10}(x)$



428A2F98 71374491 B5C0FBCF E9B5DBA5 3956C25B 59F111F1 923F82A4 AB1C5ED5
D807AA98 12835B01 243185BE 550C7DC3 72BE5D74 80DEB1FE 9BDC06A7 C19BF174
E49B69C1 EFBE4786 0FC19DC6 240CA1CC 2DE92C6F 4A7484AA 5CB0A9DC 76F988DA
983E5152 A831C66D B00327C8 BF597FC7 C6E00BF3 D5A79147 06CA6351 14292967
27B70A85 2E1B2138 4D2C6DFC 53380D13 650A7354 766A0ABB 81C2C92E 92722C85
A2BFE8A1 A81A664B C24B8B70 C76C51A3 D192E819 D6990624 F40E3585 106AA070
19A4C116 1E376C08 2748774C 34B0BCB5 391C0CB3 4ED8AA4A 5B9CCA4F 682E6FF3
748F82EE 78A5636F 84C87814 8CC70208 90BEFFFA A4506CEB BEF9A3F7 C67178F2

H_{i-1}: 256 bit I/O split into 8 words: $H_1^{(0)}, \ldots, H_1^{(7)}$

$A = H_0^{(0)} = 6A09E667$ $\quad E = H_0^{(4)} = 510E527F$

$B = H_0^{(1)} = BB67AE85$ $\quad F = H_0^{(5)} = 9B05688C$

$C = H_0^{(2)} = 3C6EF372$ $\quad G = H_0^{(6)} = 1F83D9AB$

$D = H_0^{(3)} = A54FF53A$ $\quad H = H_0^{(7)} = 5BE0CD19$