CS 5594: BLOCKCHAIN TECHNOLOGIES

Spring 2024

THANG HOANG, PhD

CRYPTOGRAPHIC PRIMITIVES

Why Cryptography?

Bitcoin is a cryptocurrency

Crypto is a mandatory building block in BTC/BC

Remark: Blockchain is based on distributed system and cryptography

Outline

Hash Functions and Applications

Asymmetric Cryptography

- Public Key Primitives
- Digital Signatures
- Elliptic Curve Cryptography

Hash Functions and Applications

Hash Function

Long message of arbitrary length → Hash → Short message of fixed length

Also known as

- Message digest
- One-way transformation
- One-way function
- Hash

$$|H(m)| \ll |m|$$

$$|H(m)| = \{160, 256, 384, 512\}$$ (preferred 256 bits)

Ideal Hash Function

Random Oracle

```
On input x ∈ {0,1}*
If x not in Book
      Flip coin d times to determine H(x)
      Record (x,H(x)) in Book
Else
      return y where (x,y) ∈ Book
EndIf
```

Impossible to build a Random Oracle in real-world

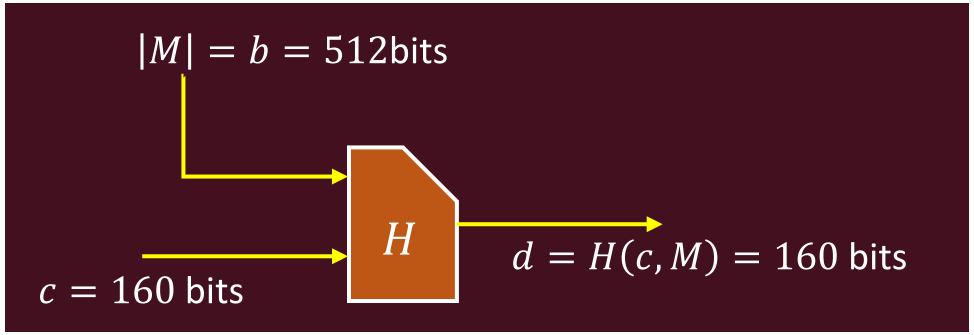
Idea: Approximate Random Oracle

Hash Function Design Principle

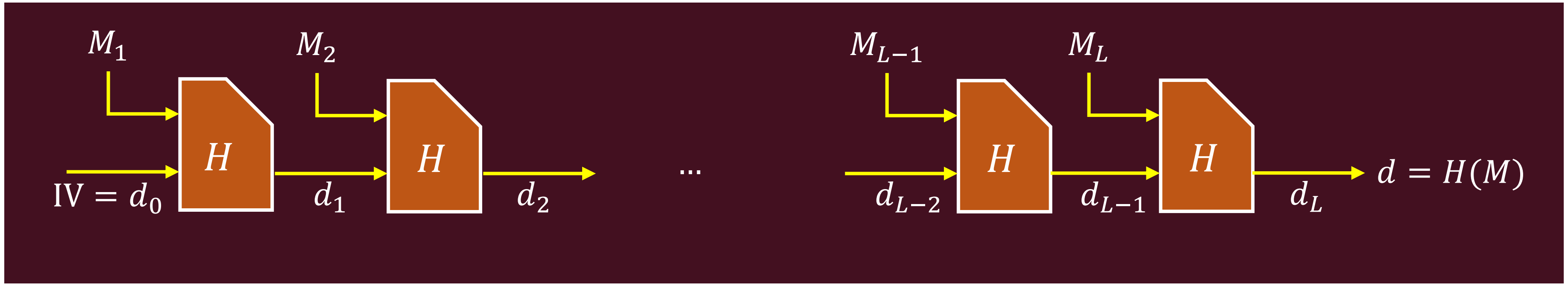
(Generally) Merkle-Damgard (MD) iteration

Start from a compression function H

$$H: \{0,1\}^{b+n} \rightarrow \{0,1\}^n$$



Iterate it



Hash-Based Applications

Create small, fixed size message digests from arbitrary message

Authentication

- Digital signatures
- Hash-based Message Authentication Codes

Pseudo Random Number Generators

Key Derivation Functions

Blockchains

Desirable Properties of Hash Functions

Performance

Easy to compute $H(m)$

One-Way (OW) property (Pre-Image Resistance)

Given $H(m)$, it is computationally infeasible to find m

$$\mathrm{m} \underset{\text{hard}}{\overset{\text{easy}}{\rightleftarrows}} H(m)$$

(Strong) Collision Resistance (CR)

Computationally infeasible to find m_1, m_2 s.t. $H(m_1) = H(m_2)$

(Weak) Collision Resistance (Target Collision Resistance – TCR)

Given $H(m)$ and m, it is computationally infeasible to find some m' s.t. $H(m') = H(m)$

CR vs. TCR

CR implies TCR

TCR does <u>not</u> imply CR

OW vs. CR

OW does <u>not</u> imply CR

CR does <u>not</u> imply OW

Length of Hash Output

Why do we need 160 or 256 bits in the output of a hash function?

If it is too long?

Unnecessary overhead

If it is too short?

Birthday paradox attack

Loss of strong collision property

Birthday Paradox

A classroom of *n* students

Probability of at least two students having the same birthday (e.g., Feb 3)

n=1 $1 - \frac{365}{365}$

n=2 $1 - \frac{364}{365}$

n=3 $1 - \left(\frac{364}{365} \times \frac{363}{365}\right)$

General *n* students:

$$1 - \left(\frac{364}{365} \times \frac{363}{365} \dots \frac{365 - n - 1}{365}\right) = 1 - \frac{365!}{(365 - n!) \cdot (365^n)}$$

Birthday Paradox

$$\Pr(\text{no collision}) = \left(1 - \frac{1}{n}\right) \times \left(1 - \frac{2}{n}\right) \times \cdots \times \left(1 - \frac{k-1}{n}\right) \approx e^{-\frac{k^2}{2n}}$$

When x is small $1 - x \approx e^{-x} \Rightarrow \left(1 - \frac{1}{n}\right) \approx e^{-\frac{1}{n}}$

$$\begin{aligned} \Pr(\text{no collision}) &= e^{-1/n} \times e^{-2/n} \times \cdots \times e^{-\frac{k-1}{n}} \\ &= e^{-((1+2+\cdots+(k-1))/n} \\ &= e^{-k(k-1)/2n} \end{aligned}$$

Birthday Paradox

$$\Pr(\text{no collision}) = e^{-k(k-1)/2n}$$

$$\Pr(\text{at least one collision}) = 1 - e^{-k(k-1)/2n}$$

$$\Rightarrow k = \sqrt{2n \times \ln(\frac{1}{1-p})}$$

In general, if there are *k* possibilities then (on average) $\sqrt{k}$ are required to find a collision

Birthday Paradox

Implication for hash function H of length m

With a probability of 0.5

If we hash $2^{m/2}$ random inputs

Two message will have the same hash output

Birthday attack

In Conclusion

Choose at least $m \geq 160$, preferably $m = 256$

Imperfect Hash Functions

Applications should rely only on "specific security properties" of hash functions

- Try to make these properties as "standard" and as weak as possible

Increase the odds of long-term security

- When weaknesses are found in hash function, application more likely to survive

- Example: MD5 is badly broken, but HMAC-MD5 is barely scratched

Security Requirements

Deterministic hashing

Attacker chooses M, $d = H(M)$

Hashing with a random salt

Attacker chooses M, then good guy chooses public salt s, $d = H(s, M)$

Hashing random messages

Given M, $d = H(M')$, where $M' = M \,||\, r$

Hashing with a secret key $\boldsymbol{k}$

Attacker chooses M, $d = H(k, M)$

Stronger

Weaker

Hashing Security Properties

Deterministic hashing

<u>Collision Resistance (CR)</u>

Attacker cannot find M, M' such that $H(M) = H(M')$

Also many other properties

Hard to find fixed-points, near-collisions, M s.t. $H(M)$ has low Hamming weight

Hashing with public salt

Target-Collision-Resistance (TCR)

Attacker chooses M, then given random salt s, cannot find M' s.t. $H(s, M) = H(s, M')$

Enhanced TCR (eTCR)

Attacker chooses M, then given random salt s, cannot find M' and s' s.t. $H(s, M) = H(s', M')$

Hashing Random Message

Second Preimage Resistance

Given random M, attacker cannot find M' such that $H(M) = H(M')$

One-wayness

Given $d = H(M)$ for random M, attacker cannot find some M' such that $H(M') = d$

Extraction

For random s, high-entropy M, the digest $d = H(s, M)$ is close to being uniform

Hashing with a Secret Key

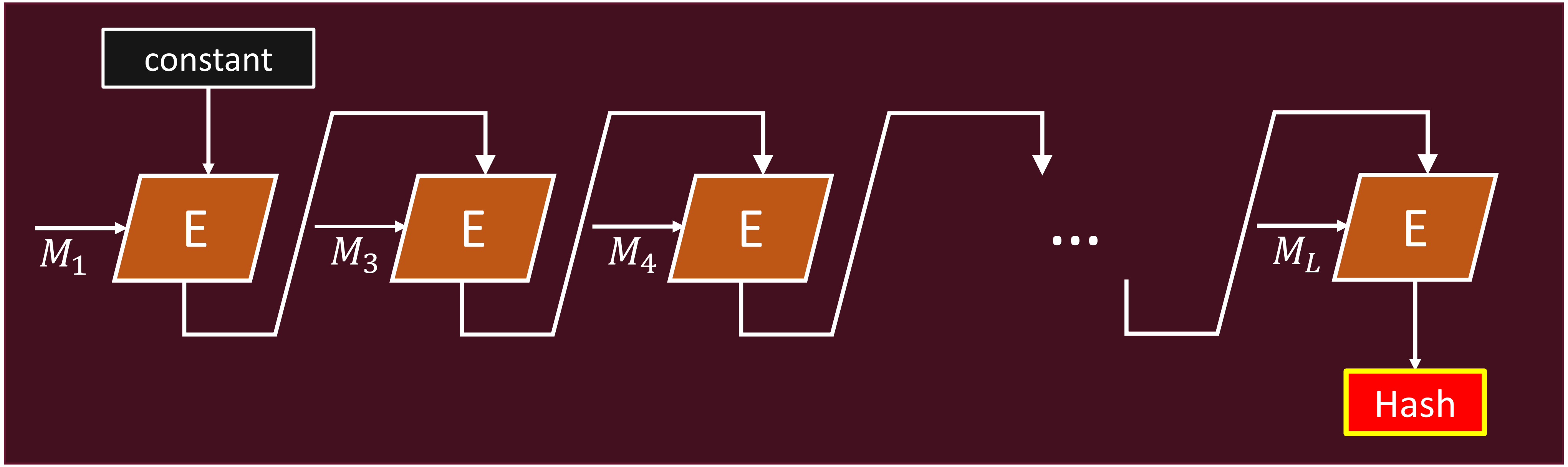
Pseudo-Random Functions

The mapping $M \rightarrow H(k, M)$ for secret key k looks random to an attacker

Universal hashing

$$\forall\, M \neq M', \Pr_k[H(k, M) = H\,(k, M')] < \epsilon$$

Is Encryption a Good Hash Function?



<u>Block-based Encryption as Hash</u>

Encryption block size may be too short (DES=64)

Birthday attack

Extension attacks

(In)Security of MD5

A few recently discovered methods can find collisions in a few hours

A few collisions were published in 2004

Many collisions found later for 1024-bit messages

More discoveries afterwards

In 2005, two X.509 certificates with different public keys and the same MD5 hash were constructed

- Based on differential analysis
- 8 hours on a 1.6GHz computer
- Much faster than birthday attack

Modern Hash Functions

MD5

- Previous versions (MD2, MD4) have serious weaknesses
- Broken; collisions published in August 2004
- Too weak to be used for critical applications

SHA-1

- Broken, both in theory and practice (with real-attacks)
- Collisions found in 2^{69} hash operations, much less than brute-force of 2^{80} operations (CRYPTO '05)
- http://thehackernews.com/2017/02/sha1-collision-attack.html

Recommended

- SHA-256, SHA-384, SHA-512, …
- BLAKE-256/512 (good for embedded devices)

Hash-based Primitives

Authentication

- Digital Signatures
- Hash-based Message Authentication Code (HMAC)
- Authenticated Data Structures
  - Hash Chain
  - Merkle Tree

Proof of Work

Digital Signatures

Hash-then-sign paradigm

First shorten arbitrary-long message, $d = H(m)$

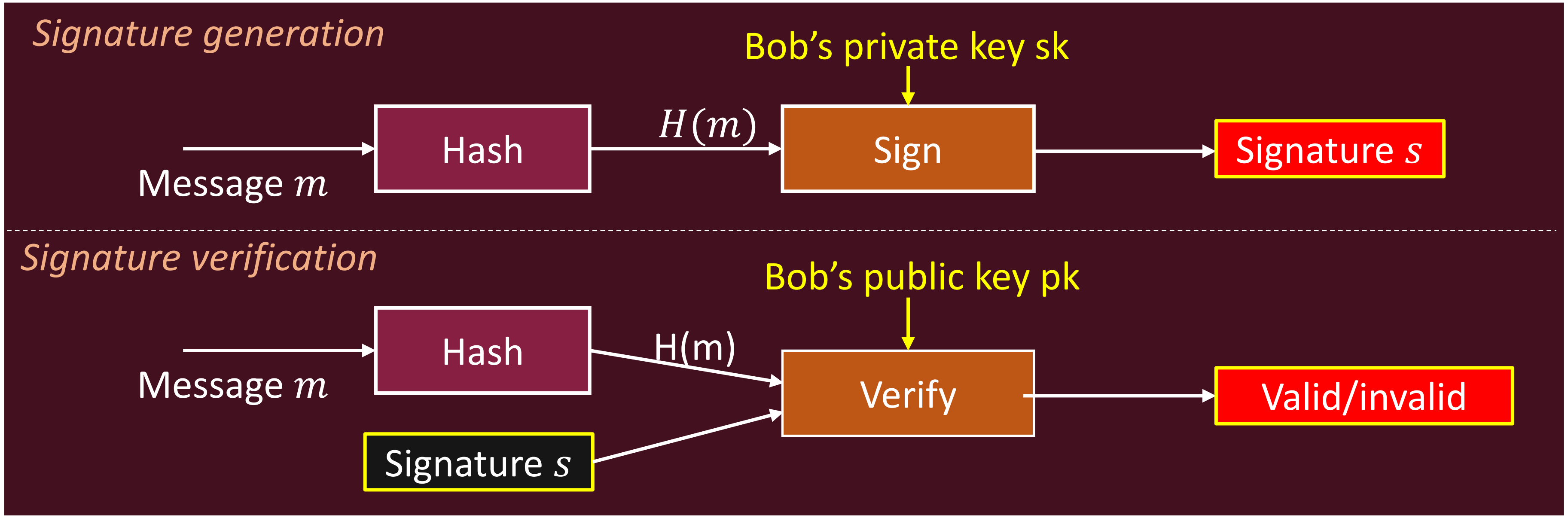
Then sign the digest, $s = \mathrm{Sign}(d)$

Rely on **collision resistance** property

If $H(M) = H(M')$ then s is a signature for both M and M'

Attacks on MD5, SHA-1 threaten current signatures

MD5 attacks can be used to get bad CA certificate [Stevens et al. 2009]



Collision Resistance is Hard

Attacker works off-line (find *M,M'*)

Use state-of-the-art cryptanalysis, as much as computation power as it can garter, without being detected !!

Helped by birthday attack (e.g., 2^80 vs. 2^160)

Well-worth the effort

One collision -> forgery for any signer

Signatures without CRHF

Use randomized hashing

To sign M, choose fresh random salt s

Set $d = H(s, M)$, then $s = \text{sign}(s, d)$

Same s (since s is explicitly signed)

Attack scenario (collision game)

Attacker chooses M, ~~M’~~

Signer chooses random salt s

Attacker mut finds $M’$ s.t. $H(s, M) = H(s, M’)$

Attack is inherently online

Only rely on **target collision resistance (TCR)**

TCR hashing for signatures

Not every randomization works

$H(M||s)$ may be subject to collision attacks when H is Merkle-Damgård

Yet this is what Probabilistic Signature Scheme (PSS) does (and it is provable in the ROM model)

Many constructions "in principle"

From any one-way function

Some engineering challenges

Most use long/variable-size randomness → don't preserve Merkle-Damgård

Also, signing salt means changing the underlying signature schemes!

Hashed Message Authentication Code (HMAC)

Simple key-prepend/append have problems when used with MD hash

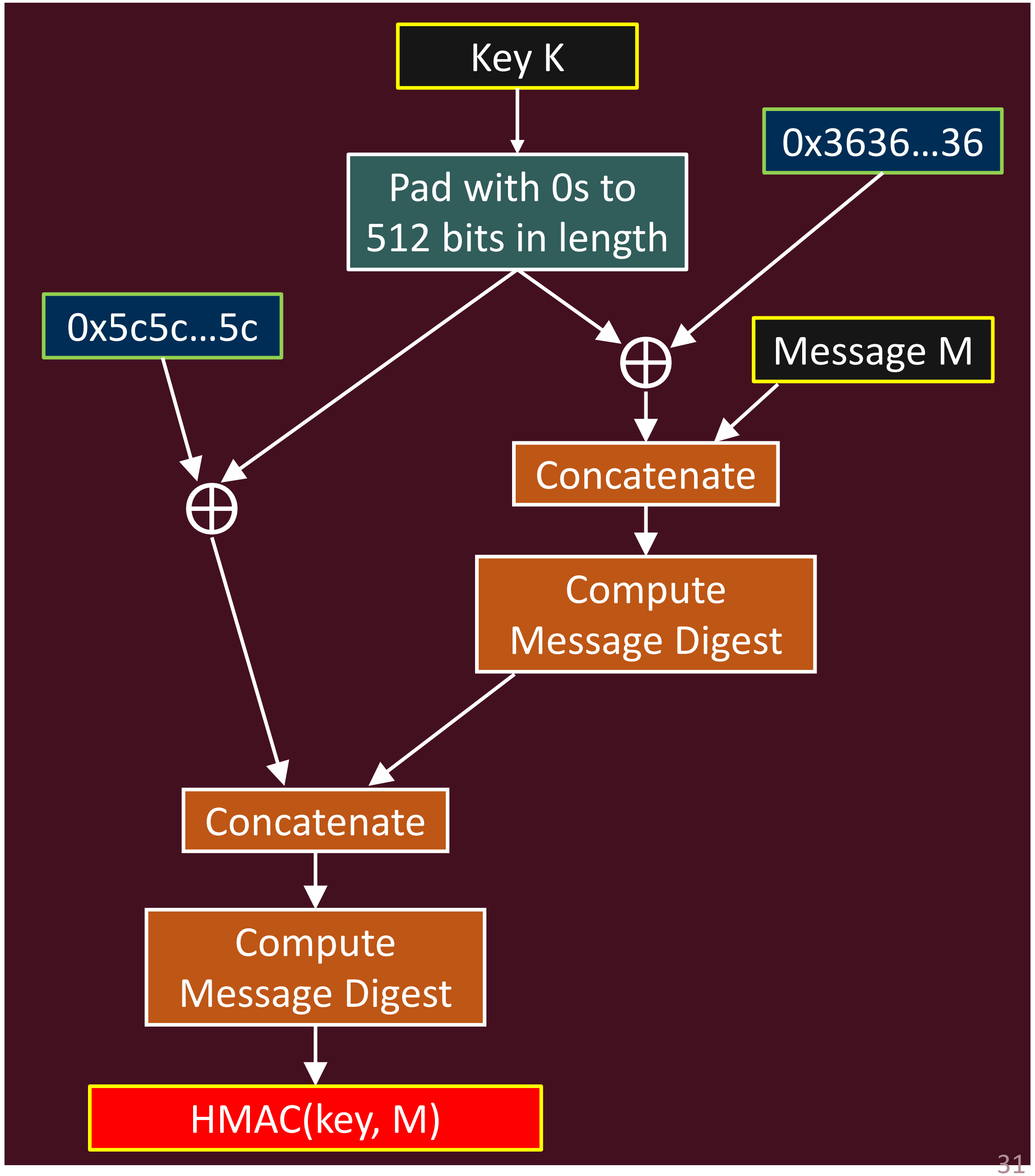
- $\text{Tag} \leftarrow H(key \,||\, M)$ subject to **length extension attacks**

HMAC: $\text{tag} \leftarrow\; H(key \,||\, H(key \,||\, M))$

- About as fast as key-prepend for a MD hash

Rely only on PRF quality of hash

- $\text{tag} \leftarrow H(key||M)$ looks random when key is secret



Hash Chain

Used for many network security applications

- S/Key
- Authenticate data streams
- Key derivation in crypto schemes
- Forward-security
- Commitments

$K_i = H(K_{i+1})$, H: hash function

Commitment

$$\boxed{K_0} \xleftarrow{F} K_1 \xleftarrow{F} K_2 \xleftarrow{F} K_3 \xleftarrow{F} K_4 \xleftarrow{F} \quad \xleftarrow{F} K_n = R$$

One-way Hash Chain: Properties

Given K_i:

- Anybody can compute K_j, where $j < i$
- It is computationally infeasible to compute K_l, where $l > i$
- Any K_l disclosed later can be authenticated by checking if $H^{l-i}(K_i) = K_l$
- Disclosing of K_{i+1} or a later value authenticates the owner of the hash chain

$$\boxed{K_0} \xleftarrow{F} K_1 \xleftarrow{F} K_2 \xleftarrow{F} K_3 \xleftarrow{F} K_4 \xleftarrow{F} \quad \xleftarrow{F} K_n = R$$

Using “Disposable” Passwords

Idea: generate a long list of passwords, use each only one time

Attacker gains little/no advantage by eavesdropping on password protocol, or cracking one password

Disadvantages

Storage overhead

Remember a lot of passwords

Alternative: the S/Key protocol

Use one-way (hash) function

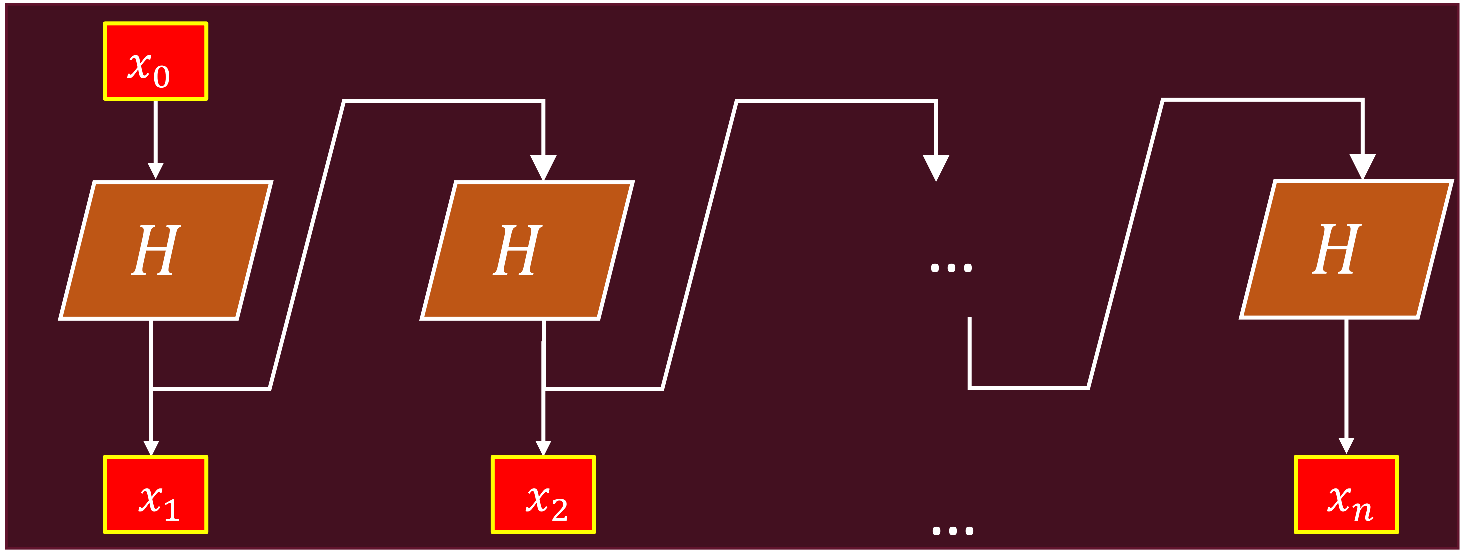
S/Key Password Generation

Alice selects a password x_0

Alice specifies n, the number of passwords to generate

Alice's generates a sequence of passwords

- $x_1 = H(x)$
- $x_2 = H(x_1)$
- …
- …
- $x_n = H(x_{n-1})$



Alice sends (securely) last value in the sequence: x_n

Key feature: no one knowing x_i can easily find x_{i-j} such that $H(x_{i-j}) = x_i, j \in [i]$

Only Alice possesses that information

S/Key Password: Limitation

Limited number of passwords (by n)

Need to periodically regenerate a new chain of passwords

Does not authenticate server!

Do not substitute bad seed password

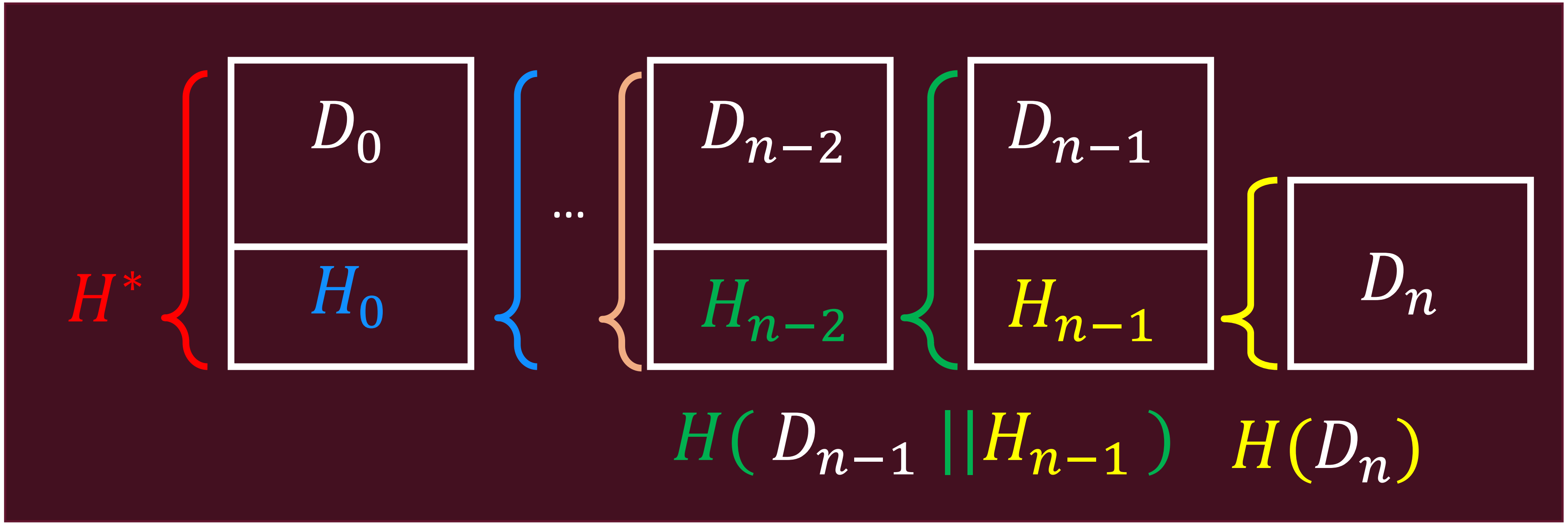
Just a tool to enhance password-based systems

Chained Hash

More general construction than one-way hash chains

Useful for authenticating a sequence of data values $D_0, D_1, \ldots, D_n$

H^* authenticates the entire chain



Merkle Hash Tree

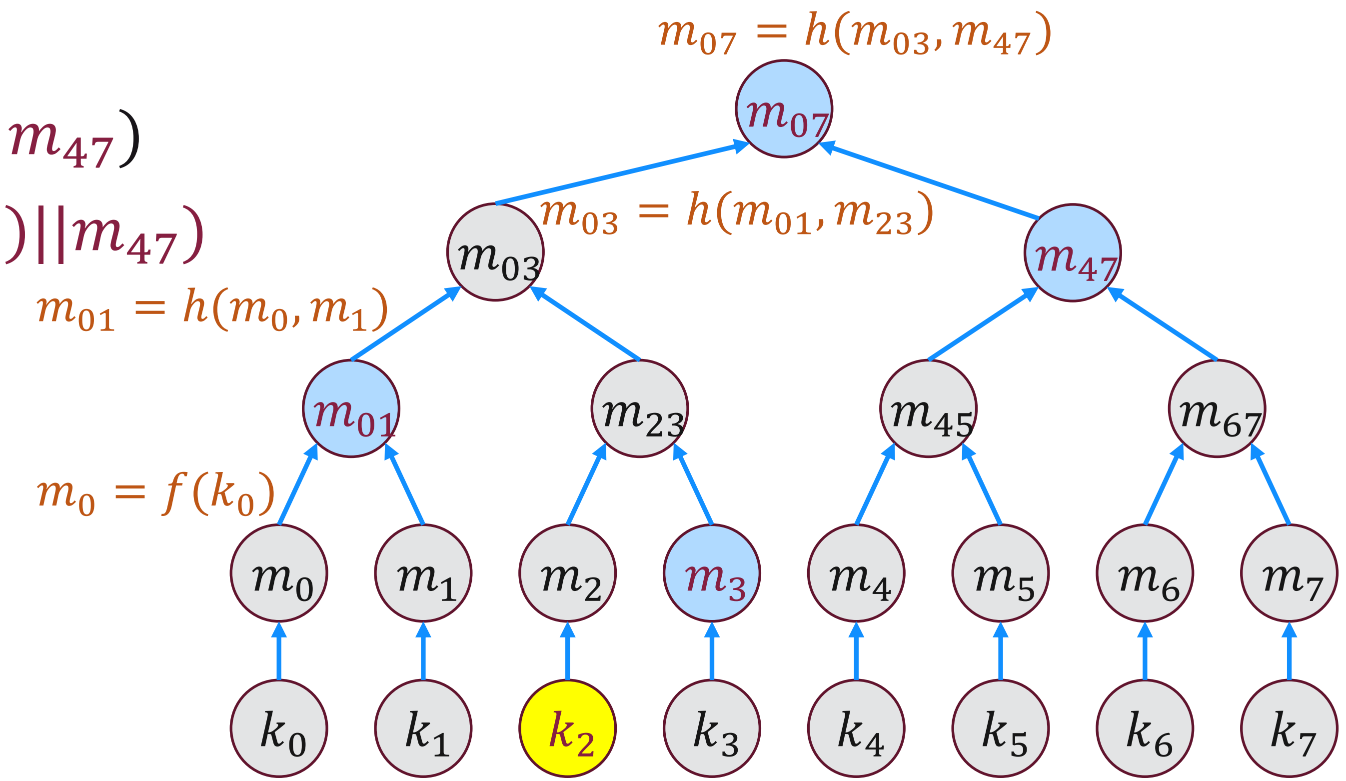
A binary tree over data values

- For authentication purpose

Verifier stores the root as the commitment of the Merkle tree

Example

- To authenticate k_2, send $(k_2, m_3, m_{01}, m_{47})$
- Check $m_{07} =? h(h(m_{01}||h(f(k_2)||m_3)||m_{47})$



Merkle Hash Tree

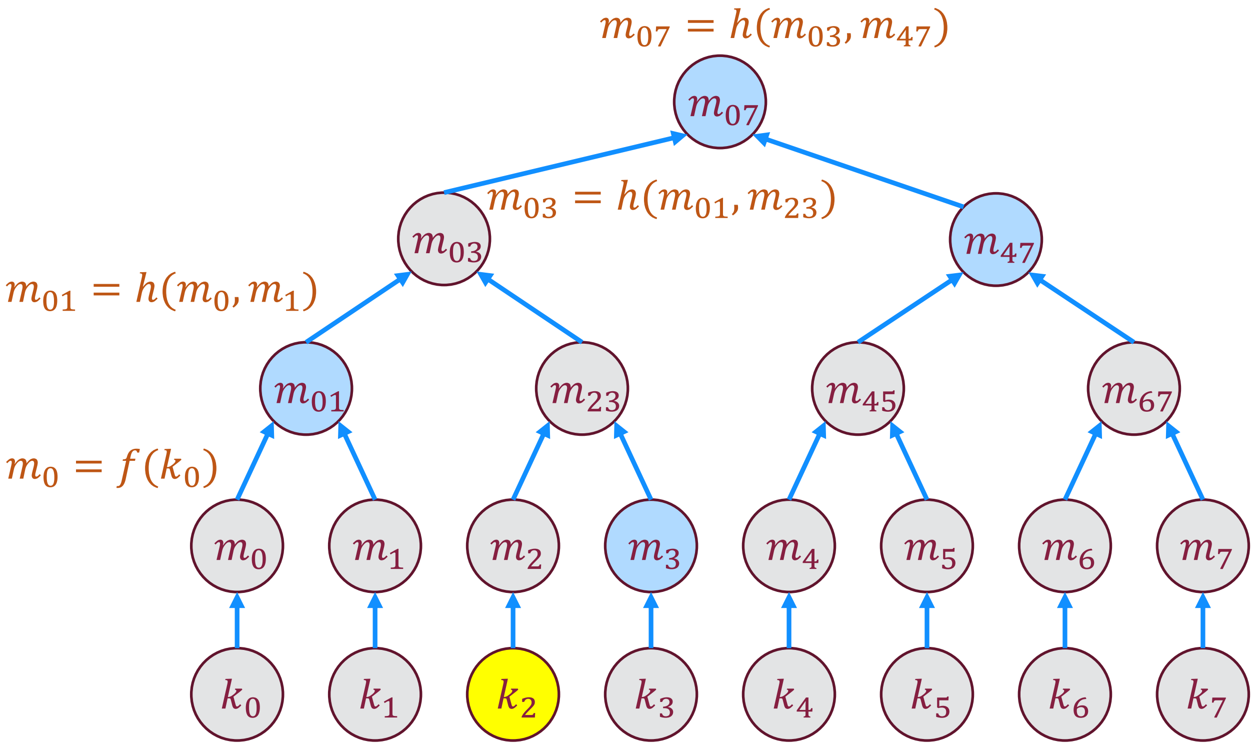
Hashing at the leaf level is mandatory to prevent unnecessary disclosure of data values

Authentication of the root is necessary to use the tree

- Typically done through a digital signature or pre-distribution

Limitation

- All leaf values must be known ahead of time

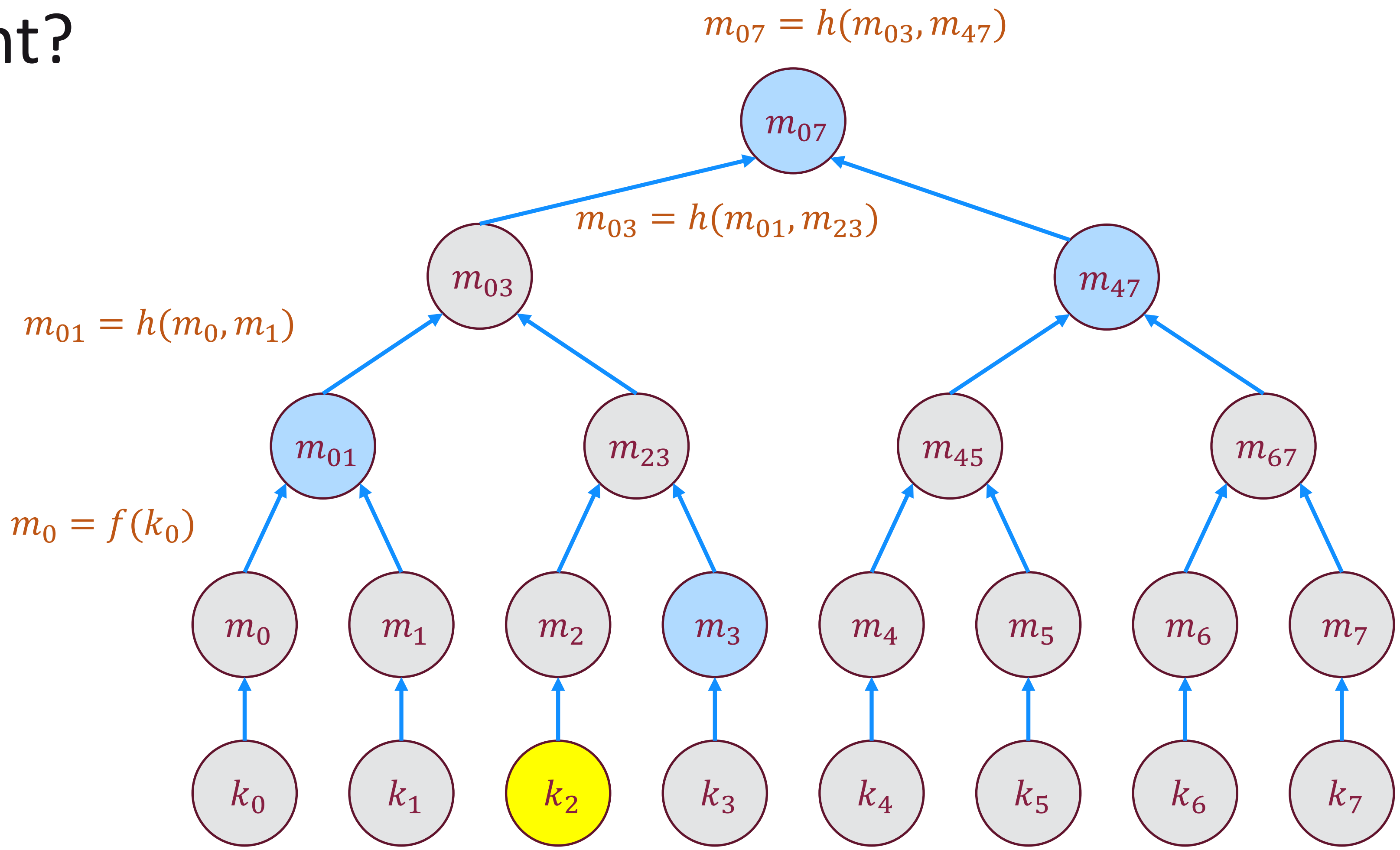


Merkle Hash Tree

Update an element?

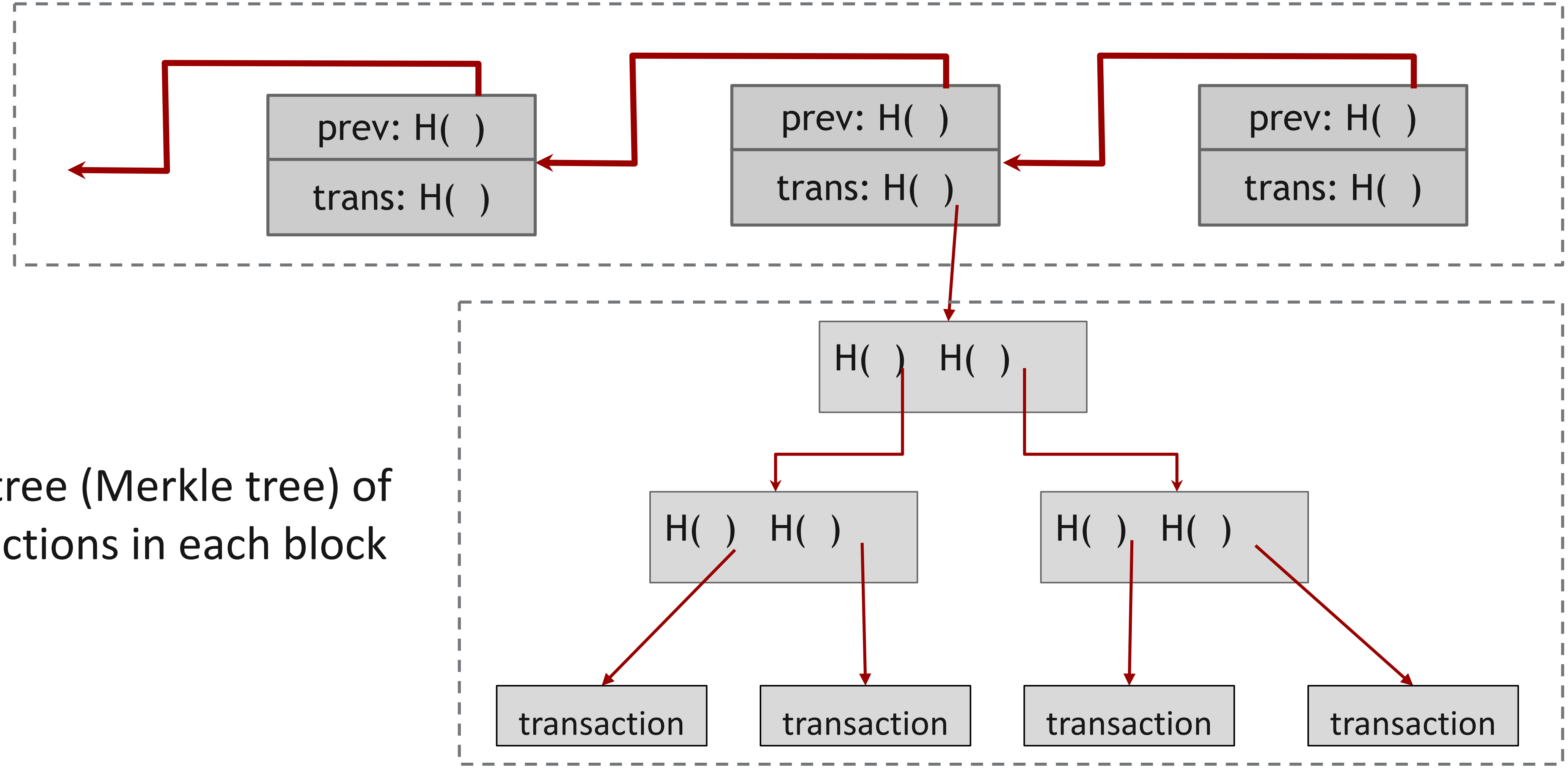
Insert a new element?

Delete an element?



Merkle Hash Tree in Blockchain

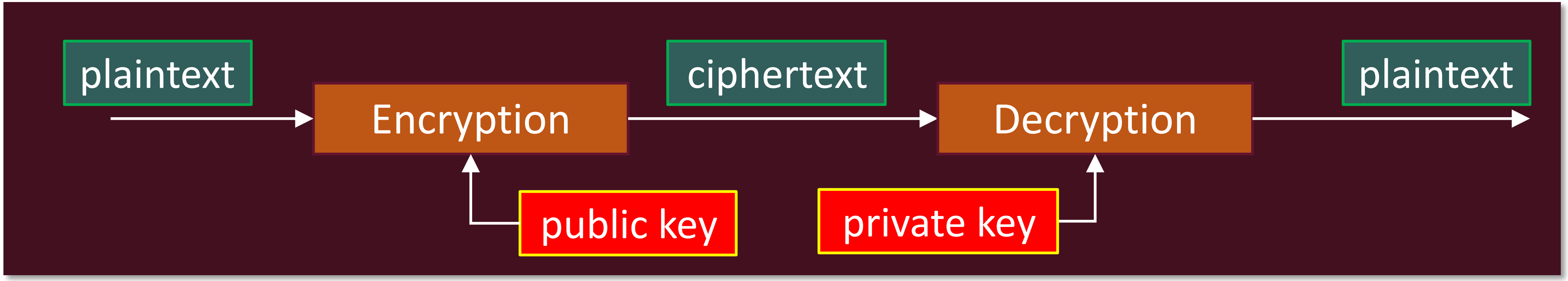
Hash chain of blocks



Hash tree (Merkle tree) of transactions in each block

Asymmetric Cryptography

Public Key (Asymmetric) Cryptography



Cryptographic operations use different keys

Known as *asymmetric key* cryptography, *public key* cryptography

- Key negotiation, digital signatures

Public Key Cryptography: Properties

Rely on some known mathematical hard problems

- Discrete logarithmic
- Elliptic curve discrete logarithmic
- Large integer factorization

P vs. NP

- A problem is in P if it can be solved in polynomial time
- A problem is in NP if the validity of a proposed solution can be checked in polynomial time

Public Key Cryptography: Properties

Confidentiality

Authentication

Integrity

<u>Non-repudiation</u>

Algebraic Structures

Public key cryptosystem harnesses certain algebraic properties in finite field

Field, Integral domain, Commutative ring, Ring, Abelian group, Group

Closure under addition	$a + b = c \in G$
Associativity of addition	$a + (b + c) = (a + b) + c$
Additive identity	$\exists e$ s.t. $a + e = e + a = a$
Additive inverse	$\exists b$ s.t. $a + b = e$
Commutativity of addition	$a + b = b + a$
Closure under multiplication	$a \times b = c \in G$
Associativity of multiplication	$a \times (b \times c) = (a \times b) \times c$
Distributive laws	$a \times (b + c) = a \times b + a \times c$
Commutativity of multiplication	$a \times b = b \times a$
Multiplicative identity	$\exists e$ s.t. $a \times e = e \times a$
No zero divisors	$a \times b = 0 \Rightarrow a = 0 \vee b = 0$
Multiplicative inverse	$\exists\, b$ s.t. $a \times b = e$

Public Key Primitives: RSA

Most popular public key method

- Provide both public key encryption and digital signatures

Operates on **multiplicative group** $(\mathbb{Z}/n\mathbb{Z})^*$

Based on factorization problem

- Given $n = p \cdot\ q$, hard to factorize n in polynomial time

Variable key length (2048 bits or greater)

Variable plaintext block size

- Plaintext block size must be smaller than key size
- Ciphertext block size is same as key size

RSA Algorithm

Find (using Miller-Rabin) large primes p and q

Let $n = p \cdot q$

Do not disclose p and q

$\Phi(n) = ???$

Choose an e that is relatively prime to $\Phi(n)$

$$\gcd(e, \Phi(n)) = 1$$

Public key = (e, n)

Find d as multiplicative inverse of $e \bmod \Phi(n)$

$$e \cdot d = 1 \bmod \Phi(n)$$

Private key = (d, n)

RSA Algorithm

Let RSA public key = (e, n) and RSA private key = (d, n)

Given a plaintext message $m < n$

Encryption

Encryption: $c \leftarrow\ m^e \bmod n$

Decryption: $m \leftarrow\ c^d \bmod n$

Signature

Signing: $s \leftarrow\ m^d \bmod n$

Verification: $m \leftarrow\ s^e \bmod n$

What if m > n?

Remark: hash-then-sign paradigm

Hashing: $t \leftarrow \text{Hash}(m)$ # |t| = 160 bits $\Longrightarrow t\ <\ n$
Signing: $s \leftarrow t^d \bmod n$

RSA Example

- Choose $p = 23, q = 11$
  - Both primes
  - $n = p \cdot q = 253$
  - $\Phi(n) = (p-1) \cdot (q-1) = 220$
- Choose $e = \mathbf{39}$
  - Relatively prime to 220
  - Public key = (**39**, 253)
- Find $d = e^{-1} \bmod 220 = \mathbf{79}$
  - $39 \cdot 79 \equiv 1 \bmod 220$
  - Private key = <**79**, 253>

Suppose plaintext $m = \mathbf{80}$

- Encryption
  $c = 80^{39} \bmod 253 = \_\_\_\_ \quad (m^e \bmod n)$
- Decryption
  $m = \_\_\_\_^{79} \bmod 253 = \mathbf{80} \quad (c^d \bmod n)$
- Signing
  $c = 80^{39} \bmod 253 = \_\_\_\_ \quad (m^e \bmod n)$
- Verification
  $m = \_\_\_\_^{79} \bmod 253 = \mathbf{80} \quad (c^d \bmod n)$

RSA Example

- Choose $p = 23, q = 11$
  - Both primes
  - $n = p \cdot q = 253$
  - $\Phi(n) = (p-1) \cdot (q-1) = 220$
- Choose $e = \mathbf{39}$
  - Relatively prime to 220
  - Public key = (**39**, 253)
- Find $d = e^{-1} \bmod 220 = \mathbf{79}$
  - $39 \cdot 79 \equiv 1 \bmod 220$
  - Private key = <**79**, 253>

Suppose plaintext $m = \mathbf{80}$

- Encryption
  $c = 80^{39} \bmod 253 = \mathbf{37}$ $(m^e \bmod n)$
- Decryption
  $m = \mathbf{37}^{79} \bmod 253 = \mathbf{80}$ $(c^d \bmod n)$
- Signing
  $s = 80^{79} \bmod 253 = \mathbf{224}$ $(m^d \bmod n)$
- Verification
  $m = 224^{39} \bmod 253 = \mathbf{80}$ $(s^e \bmod n)$

RSA Security

At present, 1024-bit keys are considered secure, but 2048-bit keys are recommended

Tips for making n hard to be factorized

- p and q lengths should be similar (e.g., ~500 bits each if key is 1024 bits)
- both $(p-1)$ and $(q-1)$ should contain a "large" prime factor
- $\gcd(p-1, q-1)$ should be "small"
- d should be larger than $n^{0.25}$

RSA Security

Some attacks on RSA

- Mathematical attacks (factor n, compute d from e) -> extremely difficult
- Brute force
- Probable-message attacks
- Timing attacks

How to prevent attacks?

- Large key
- Random padding (OKCS #1 v1)
- Message blinding

Digital Signature Algorithm (DSA)

Useful only for digital signing (no encryption or key exchange)

Components

- SHA-1 to generate a hash value (some other hash functions also allowed now)
- Digital Signature Algorithm (DSA) to generate the digital signature from this hash value

Designed to be fast for the signer rather than verifier

Based on discrete log hard problem

- Given y_M , hard to find x_M s.t. $y_M = g^{x_M} \bmod p$

DSA Public Parameters

Announce public parameters used for signing

Pick p as a prime with >= 1024 bits

$p = 103$

Pick q as a 160-bit prime such that $q|(p–1)$

$q = 17$ (divides 102)

Choose $g \equiv h^{(p-1)/q} \bmod p$,

where $1 < h < (p - 1)$ such that $g > 1$

If $h = 2$, $g = 2^6 \bmod 103 = 64$

powers of 64 mod 103 =
64 79 9 61 93 81 34 13 8 100 14 72 76 23 30 66 1

17 values

Note: g is of order $q \bmod p$

DSA Key Gen and Signing

Key Generation

- Alice generates a long-term private key x_M
  - Random integer $0 < x_M < q$

$x_M = 13$

- Alice generates a long-term public key y_M
  - $y_M = g^{x_M} \bmod p$

$y_M = 64^{13} \bmod 103 = 76$

- Alice randomly picks a private key k such that $0 < k < q$, and generates $k^{-1} \bmod q$

$k = 12;\ k^{-1} = 12^{-1} \bmod 17 = 10$

Signing phase

- Signing message M
  - Public key $r = (g^k \bmod p) \bmod q$
  - Signature $s = \left(k^{-1}(H(M) + x_m \cdot r)\right) \bmod q$
- Send (M, r, s)

$H(M) = 75$

$r = (6412 \bmod 103) \bmod 17 = 4$

$s = \left(10 \cdot (75 + 13 \cdot 4)\right) \bmod 17 = 12$

$(M, 4, 12)$

DSA Verification

Verification

- Public parameters: g, p, q, y_M

$p = 103, q = 17, g = 64, y_M = 76$

- Received from signer: M, r, s

$\boxed{M, 4, 12}$ $\boxed{H(M) = 75}$

- $w = (s)^{-1} \bmod q$

$\boxed{w = 12^{-1} \bmod 17 = 10}$

- $u_1 = [H(M)w] \bmod q$

$\boxed{u_1 = 75 \cdot 10 \bmod 17 = 2}$

- $u_2 = (r * w) \bmod q$

$\boxed{u_2 = 4 \cdot 10 \bmod 17 = 6}$

- $v = [(g^{u_1} \cdot y_M^{u_2}) \bmod p] \bmod q$

$\boxed{v = ((64^2 \cdot 76^6) \bmod 103) \bmod 17 = 4}$

1. If $v = r$, then the signature is verified

DSA Security

Given y_M, it is difficult to compute x_M

x_M is the discrete log of y_M to the base g, mod p (i.e., $y_M = g^{x_M} \bmod p$)

Similarly, given r, it is difficult to compute k

Cannot forge a signature without x_M

Signatures are not repeated (used once per message) and cannot be replayed

Slower to verify than RSA, but faster signing than RSA

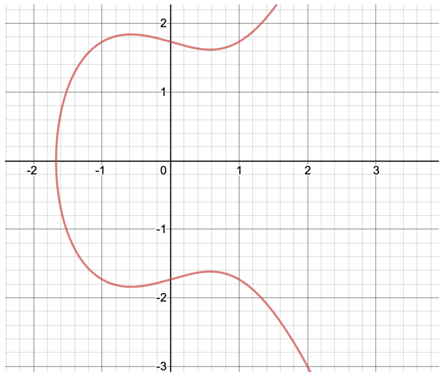
Key lengths of 2048 bits and greater are also allowed

Elliptic Curve Cryptography

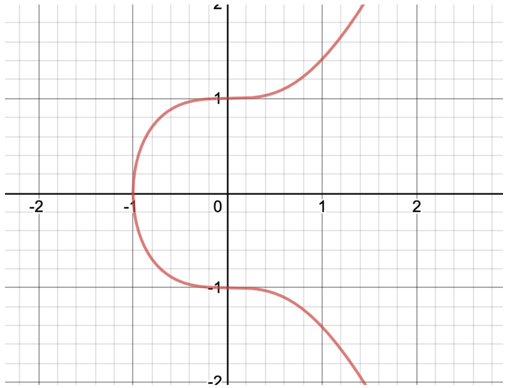
An elliptic curve (EC) consists of all elements $(x, y) \in \mathbb{F}$ satisfying

$$y^2 = x^3 + ax + b$$

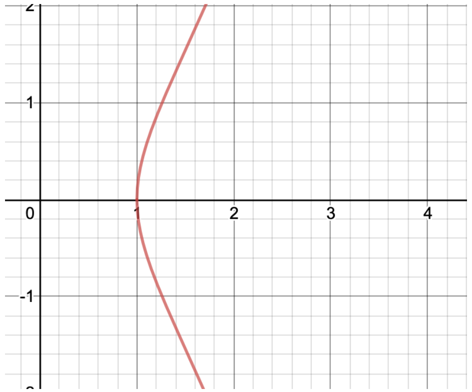
$y^2 = x^3 - x + 3$



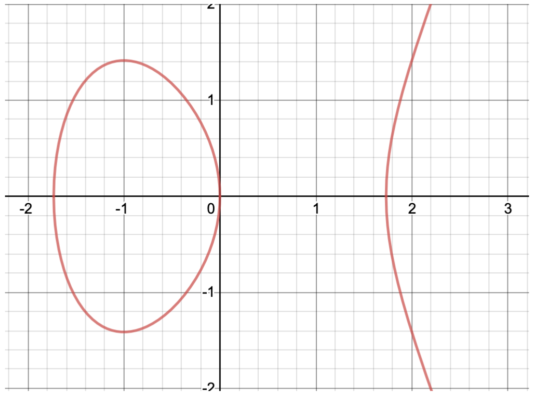
$y^2 = x^3 + 1$



$y^2 = x^3 - 1$



$y^2 = x^3 - 4x$

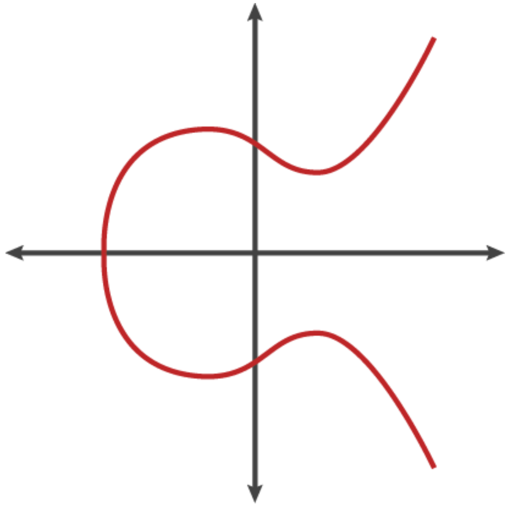


Why Elliptic Curve Cryptography?

Shorter key size than conventional PKCs (DL-based, RSA)

Lower computation overhead

Due to shorter key



Sec level (bits)	RSA/DL-based key size (bits)	ECC key size (bits)
56	512	112
80	1024	160
112	2048	224
128	3072	256
192	7680	384
256	15360	512

Elliptic Curve Cryptography

- **Point addition:** Let P and Q be two EC points

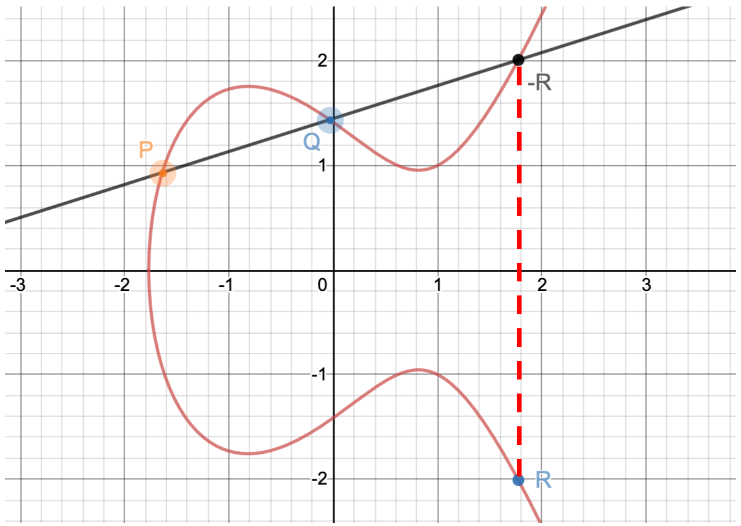
$$P + Q = R = (x, -y),$$

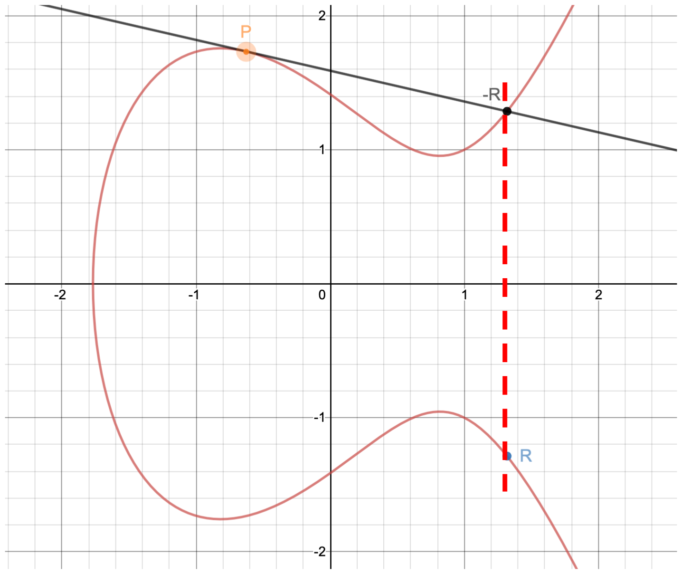
$$(x, y) = -R := \text{intersection of EC and PQ-line}$$

- **Point negation:** $P + (-P) = O$
  - *O: identity point at infinity (not on the curve)*
- **Point doubling:** $R = P + P = (x', -y'),$

$$(x', y') = -R := \text{intersection of EC and tangent line of P}$$

- **Point multiplication:** achieved via double-and-add
  - Similar to multiply-and-square trick
  - e.g., Q=7P, 7 = $(111)_2$, Q = 0, R=P
    - Q += R & R*=2; Q+=R & R*=2; Q+=R & R*=2





Point addition and point doubling (arithmetic)

$$x_3 = s^2 - x_1 - x_2$$
$$y_3 = s(x_1 - x_2) - y_1$$

where

$$s = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \text{ if } P \neq Q \text{ (point addition)} \\ \dfrac{3x_1^2 + a}{2y_1} \text{ if } P = Q \text{ (point doubling)} \end{cases}$$

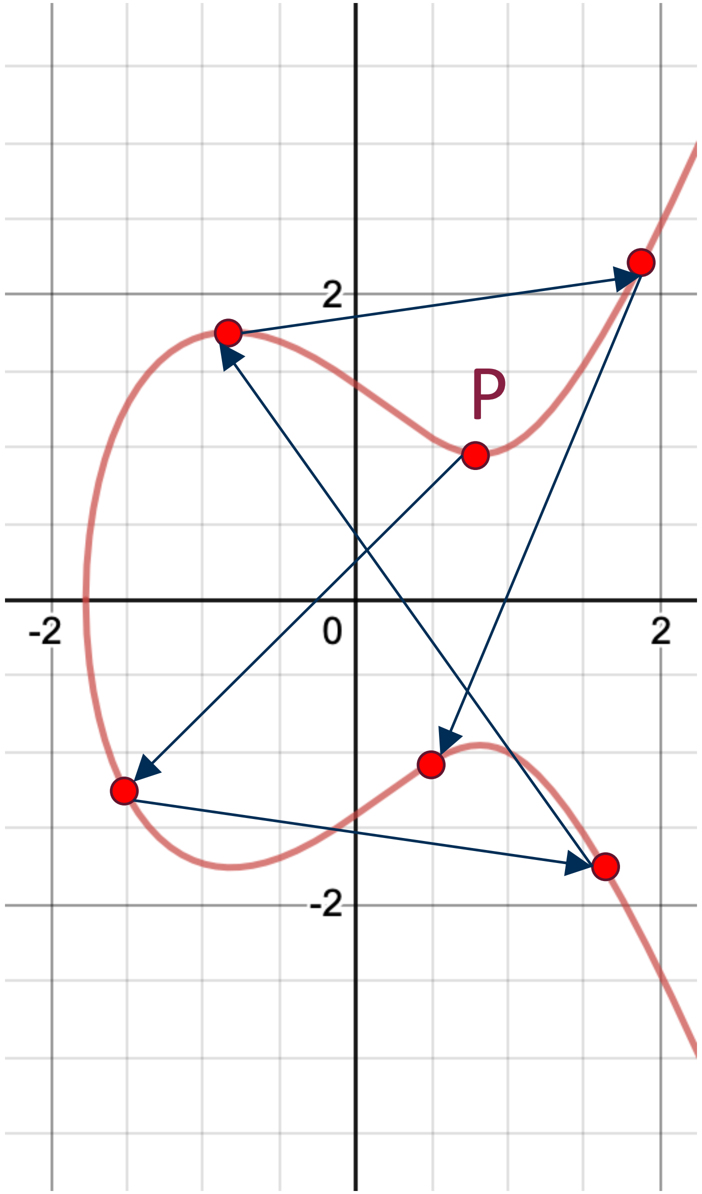
- Example: Let $\mathrm{EC} = y^2 = x^3 + 2x + 2 \bmod 17$, $\mathrm{P} = (5,1), \mathrm{Q} = (7,6)$
- Compute $\mathrm{U} = 2P, V = P + Q$
  - $s_u = \frac{3x_1^2+a}{2y_1} = (3 \cdot \_\_ + 2) \cdot (2 \cdot \_\_)^{-1} = \_\_ \cdot \_\_^{-1} = \_\_ \cdot \_\_ \equiv \_\_ \bmod 17$
  - $s_v = \frac{y_2-y_1}{x_2-x_1} = (\_\_ - \_\_) \cdot (\_\_ - \_\_)^{-1} = \_\_ \cdot \_\_^{-1} = \_\_ \equiv \_\_ \bmod 17$
  - $x_u = s_u^2 - x_p - x_q = \_\_ \bmod 17;\ y_u = s_u(x_p - x_q) - y_p = \_\_ \bmod 17$
  - $x_v = s_v^2 - x_p - x_q = \_\_ \bmod 17;\ y_v = s_v(x_p - x_q) - y_p = \_\_ \bmod 17$

Points on an elliptic curve and the infinity point O form cyclic subgroups

e.g., $y^2 = x^3 + 2x + 2 \bmod 17$, P $=$ $(5,1)$

2P = (6,3); 3P = 2P+P = (10,6); ……., 18P = (5,16); 19P = O

EC has order |E| = 19 as there are 19 points in its cyclic group



How many points in an arbitrary EC?

Given an elliptic curve E modulo p, the number of points on E is bounded by

$p + 1 - 2\sqrt{p} \leq |E| \leq p + 1 + 2\sqrt{p}$ (**Hasse Theorem**)

Number of points close to prime p

Elliptic Curve Cryptography

Rely on EC-discrete logarithmic hard problem

Given $(G, Y) \in \mathrm{EC}$ s.t. $Y = k \cdot G$ (Y is G added to itself k times), hard to find k

EC Key size smaller than RSA/DH-based crypto

Attacks on EC groups are weaker than factorizing algorithm or discrete log attacks

Best known attacks

Baby-step, giant step

Pollard-Rho

Number of trials: $\mathrm{O}(\sqrt{p})$

ECDSA Public Parameters

Public parameter generation

Pick p as a prime with >= 160 bits

$p = 17$

Pick a, b to form an EC

$y^2 = x^3 + 2x^2 + 2x$
(a=2,b=2)

Pick an ECC generator G with order q

$$q \times G = 0$$

G = (5,1), q=19

How to choose G and q?

multiplication of G mod p =
(5,1) (6,3) (10,6) (3,1) (9,16) (16,13) (0,6) (13,7) (7,6) (7,11) (13,10) (0,11) (16,4) (9,1) (3,16) (10,11) (6,14) (5,16) (O)

19 points

(p,a,b,G,q) are public parameters

ECDSA Key Gen and Signing

Key Generation

Alice generates a long-term private key d_A

Random integer $0 < d_A < q$

$\boxed{d_M = 5}$

Alice generates a long-term public key Q_A

$Q_A = d_M \times G \bmod p$

$\boxed{Q_A = (9,16)}$

Signing phase: To sign message M

$\boxed{z = H(M) = 5}$

Select an ephemeral key k from $[1, q-1]$

$\boxed{k = 3}$

Compute an EC point $(x_1, y_1) = k \times G$

$\boxed{(x_1, y_1) = (10,6)}$

Compute $r = x_1 \bmod q$ (choose other k if $r = 0$)

$\boxed{r = 10}$

Compute $s = k^{-1}(z + r \cdot d_A) \bmod q$ (choose other k if $s = 0$)

Signature $\sigma = (r, s)$

$\boxed{s = 13 \cdot (5 + 10 \cdot 5) \bmod 19 = 12}$

Send (M, σ)

$\boxed{(M, 10, 12)}$

ECDSA Verification

Verification

Public parameters: a, b, G, q Q_A

$a = 2, b = 2, p = 17, q = 19, G = (5,1), Q_A = (9,16)$

Received from signer: M, r, s

$M, 10, 12$

$z = H(M) = 5$

$$u_1 = z \cdot s^{-1} \bmod q$$

$$u_1 = 5 \cdot 12^{-1} \bmod 19 = 2$$

$$u_2 = r \cdot s^{-1} \bmod q$$

$$u_1 = 10 \cdot 12^{-1} \bmod 19 = 4$$

Compute EC point $(x_1, y_1) = u_1 \times G + u_2 \times Q_A$

$$(x_1, y_1) = (6,3) + (5,1) = (10,6)$$

If $(x_1, y_1) = O$, invalid signature

$$x_1 = 10, r = 10$$

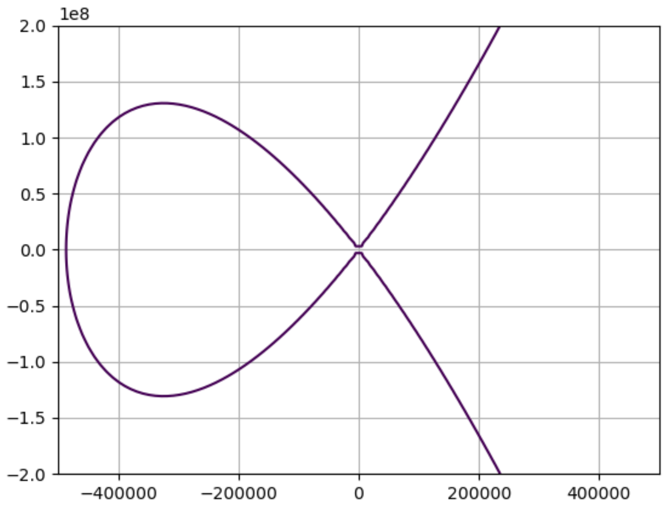
If $r \equiv x_1 \bmod n$, valid signature. Invalid otherwise

Some Popular ECs

Curve25519 (Montgomery curve)

$$y^2 = x^3 + 486662x^2 + x$$

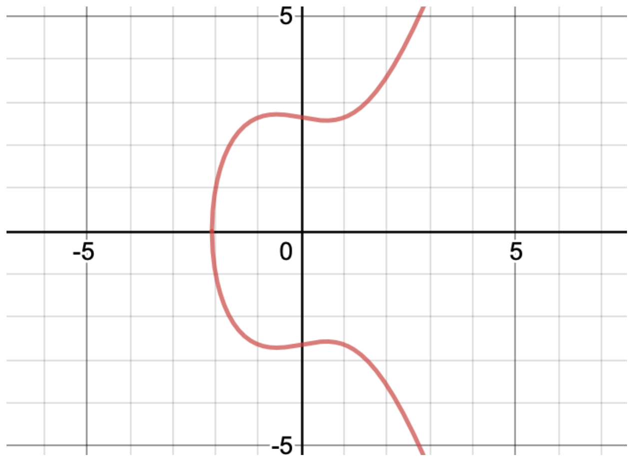
$$p = 2^{255} - 19$$



Secp256k1 (used in bitcoin)

$$y^2 = x^3 + 7$$

$$p = 2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$$



Other ECC-based Primitives

ECC replaces modular arithmetic operations in conventional PKC by operations defined over the elliptic curve

ECC primitives can be easily constructed by making analogous changes to the corresponding conventional PKC

- ECC Encryption from ElGamal Encryption
- ECC-DH Key Exchange from Diffie-Hellman Key Exchange
- ECC-DSA Signature from DSA Signature