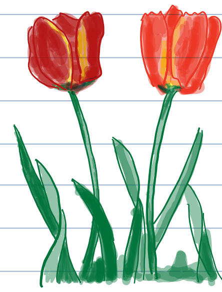
Secure Distributed Systems
CompSci 661 / 461



This video

- Overview of applied cryptography
    - Kerckchoff's Principle
    - Symmetric/Asymmetric ciphers
    - Cryptographic hash algorithms
    - Merkle Trees
    - Diffie-Helman Key Exchange

Cryptography allows two parties to communicate messages to each other securely despite an eavesdropper

- Alice and Bob will be our two parties
- Alice wants to send plaintext p
  - She instead sends ciphertext c, which has been encrypted with key k
  - It must be that k was previously shared between Alice and Bob securely
  - The key is a shared secret.

- A cryptographic system (an algorithm, a cipher) input p and k, and it produces c.
  - producing c from p with k is computationally easy.
  - and p from c with k is computationally easy.
  - But producing p from c without k shoud be computationally infeasible.

> **Kerckhoff's Principle** (1883)
> A cryptosystem should be secure even if the attacker knows all the details of the system, with the exception of the secret key.

Modern ciphers include

Symmetric ciphers:

same key for encryption and decryption

AES advanced encryption standard

- previously DES was the standard

K_{AB} — a symmetric key shared between Alice and Bob

Therefore we may write

$C = \{P\}_{K_{AB}}$ — C is the encrypted value of P

$P = \{C\}_{K_{AB}}$ — decrypt C and we get P back

Asymmetric ciphers:

two keys — ciphertext generated with one key can be decrypted to plaintext by the other key only.

e.g., RSA and Elliptic Curve Crypto (ECC)

Typically, operate 2-3 orders of magnitude slower than symmetric key crypto algs.

Generally, in asymmetric systems, a participant Alice generates two keys

K_{A-} her (secret) private key

K_{A+} her public key

Therefore we may write

$$C = \{P\}_{K_{A+}}$$

$$P = \{C\}_{K_{A-}}$$

C is the encrypted value of P. But here, we encrypt with one key and decrypt only with the other.

Ideally, given a crypto system, an attacker could recover P from C only by trying every possible key. (brute force)

- if the space of all possible keys is n bits, then the number of operations required by the attacker is $O(2^n)$
  - on average, the attacker will succeed after trying half the keys.

Typically, there is some attack which allows the attacker to do better than brute force and $O(2^n)$

To compensate, the key space is made larger.

- Hence RSA key lengths are larger than ECC.
- But longer keys implies that the execution of encryption is also longer duration:

Attacks on RSA involve factoring the product of two large prime numbers.

Attacks on Elliptic curves involve solving the discrete logarithm problem. (explained later)

Algorithm Family	Cryptosystems	Security Level (bit) 80	128	192	256
Integer factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete logarithm	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Symmetric-key	AES, 3DES	80 bit	128 bit	192 bit	256 bit

Table by C. Paar

Bitcoin uses 256-bit ECDSA keys.

Cryptographic Hash Functions

- One-way functions
- Input can be of any size
- Output is a (small) fixed length
  e.g., 160 bits
- Typically, 2–3 orders of magnitude faster than symmetric key crypto

- If $h(\cdot)$ is our hash function, then

$$z = h(x)$$

z is the resulting "message digest" (or "hash")
x is called the "pre-image" of z

With a good hash function, you can expect that changing just one bit of the pre-image will flip each bit of the output with ½ probability

There are several critical properties of such functions that make them cryptographically strong.

① Pre-image resistance (one-wayness)

Given hash output z, it must be computationally infeasible to find an input message x such that $z = h(x)$

② Second pre-image resistance (weak collision)

Given x_1 and $h(x_1)$, it should be computationally infeasible to find an $x_2 \neq x_1$ such that $h(x_2) = h(x_1)$

③ Collision Resistance (Birthday Attack)

It should be computationally infeasible to find any $x_1 \neq x_2$ such that $h(x_1) = h(x_2)$

Example algorithms include

MD-5, SHA-1, SHA-256, SHA-512

Applications of hash functions

① Equivalence of two documents

If two documents x_1 and x_2 are the same, with even a single bit different, then the following is true:

$$h(x_1) = h(x_2)$$

② Equivalence of Ordered Sets

$$P = p_1, p_2, \dots, p_n$$
$$Q = q_1, q_2, \dots, q_n$$

P and Q are the same if $h(P) = h(Q)$

This does not work for unordered sets (unless you order them first).

3 One-time passwords ("S/Key")

p = master password

t = counter

Let $p_1 = h(p)$

$p_2 = h(p_1) = h(h(p))$

$p_i = h(p_{i-1})$

- Alice sets up the system to store $t=100$ and P_{100}.
- To log in, she is presented with a challenge of $t=99$.
- She sends P_{99}.
- The system checks if $h(P_{99}) = P_{100}$
- If so, access is granted and P_{99} is stored and the counter is decremented $t=99$.

The next time she logs in, she's presented with $t=98$ as the challenge.

Then $t=97$...

Eventually, we need to re-initialize

④ Sign a message m

A cryptographic signature allows the owner of a key to authicate a document, such that others can verify. If the document changes, the signature won't validate.

To sign, encrypt the hash of the document

$$\{h(m)\}_{K_{A-}} = s$$

private key

- to verify

let $z = h(m)$

let $p = \{s\}_{K_{A+}}$ public key

if $z = p$, then the signature is valid

Sometimes I use sq. brackets to denote signing.

$$[m]_{K_{A-}}$$

5 Merkle Trees (Ralph Merkle)

Check if an item is in a ordered set.

(Why ordered?)

Let P be an ordered set

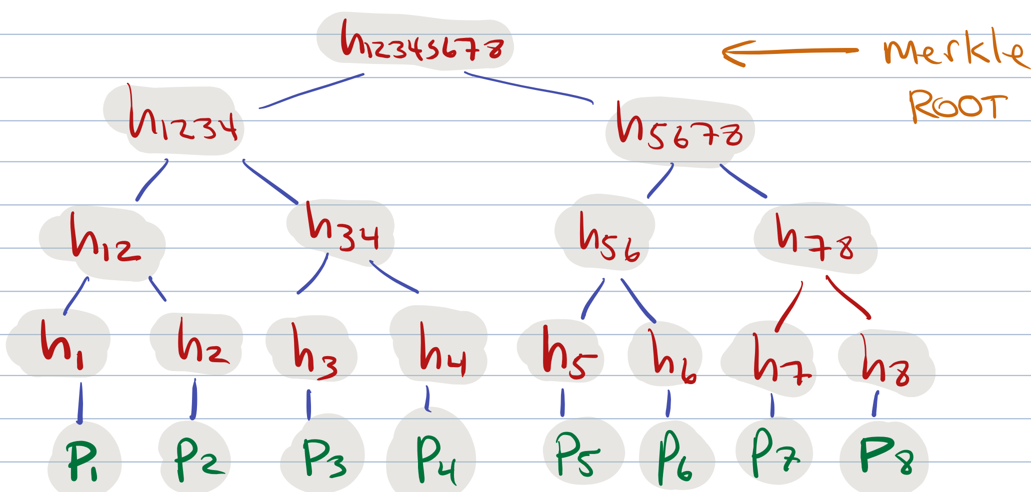
$$P = P_1, P_2, \ldots, P_8$$

Set up a Tree such that

- each element of P is a leaf
- each leaf p_i has a parent $h_i = h(p_i)$
- pairs of parents are given parent

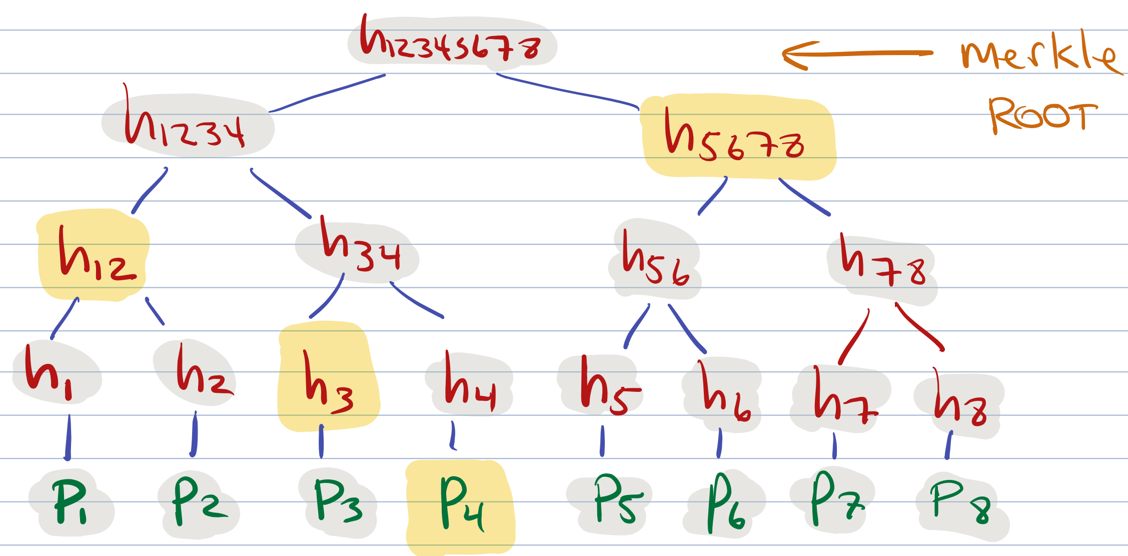
$$h_{ij} = h(h_i, h_j)$$

concatenation



- The entire set P is represented by the Merkle root.

- To prove an element is in the set, we must provide $\log_2(|P|)$ values from the inner tree, where $|P|$ is the size of P:



To add and subtract values, we do not need to rehash every element in P.

Another way to check if an item is in a set?

- Just hash the entire set.
- Ask youself, what are the advantages of the Merkle Tree method?

Something else we'll need:

- A nonce is a random value never used again.
- The current time is a nonce, but often we'll need more than one per smallest unit of time
- Often it's a large random number since the chance of repeats are very, very small.
- Sometimes it's the time concatenated with a random number. But be careful, it's only the random number that is unguessable. The time makes it easier for us to not have to keep track of all numbers used ever.

Diffie-Hellman Key Exchange

I Setup

1. Choose a large prime p
2. Choose an integer $\alpha \in \{2, 3, \ldots, p-2\}$
3. Publish p and α.

Assume that Alice & Bob know p and α correctly

II KEY Exchange

1. Alice : chooses $a = K_{A-} \in \{2, \ldots, p-2\}$ private
   compute $A = K_{A+} = \alpha^a \bmod p$ public

2. Bob : choose $b = K_{B-} \in \{2, \ldots, p-2\}$ private
   compute $B = K_{B+} = \alpha^b \bmod p$ public

3. $A \to B : A$ Alice sends A to Bob
   $B \to A : B$ Bob sends B to Alice

4. Alice computes their shared key

$$K_{AB} = B^a \bmod p = (\alpha^b)^a \bmod p = \alpha^{ab} \bmod p$$

Bob computes their shared key.

$$K_{AB} = A^b \bmod p = (\alpha^a)^b \bmod p = \alpha^{ab} \bmod p$$

Now Alice and Bob have the same shared key.
They use it with, for example, AES to exchange encrypted data.

Why is DHKE Secure?

The short answer?

When Alice sends A over the network, an eavesdropper cannot determine a from $\alpha^a \bmod p$.

That is, exponentiation in $\bmod p$ is a one-way function.

- even though α and p are public.

Recall that $z = x^y$ then $\log_x(z) = y$

for example $2^3 = 8$ and $\log_2(8) = 3$

Here we have $A = \alpha^a \bmod p$

and $a = \log_\alpha(A) \bmod p$

But taking the logarithm is computationally infeasible.

For the same reason, given

$$K_{AB} = \alpha^{ab} \bmod p$$

it's not feasible to compute

$$ab = \log_\alpha(K_{AB}) \bmod p$$

This computation is called the discrete logarithm problem.

It increases in difficulty with our prime p.