Public Key Crypto

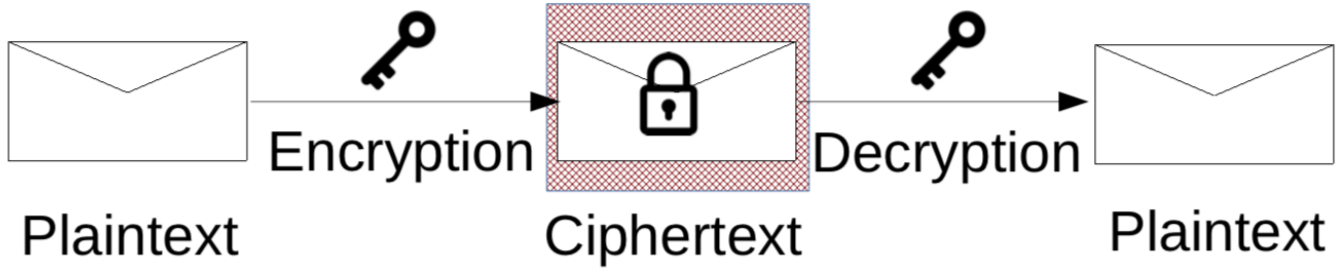
Shuai Wang

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

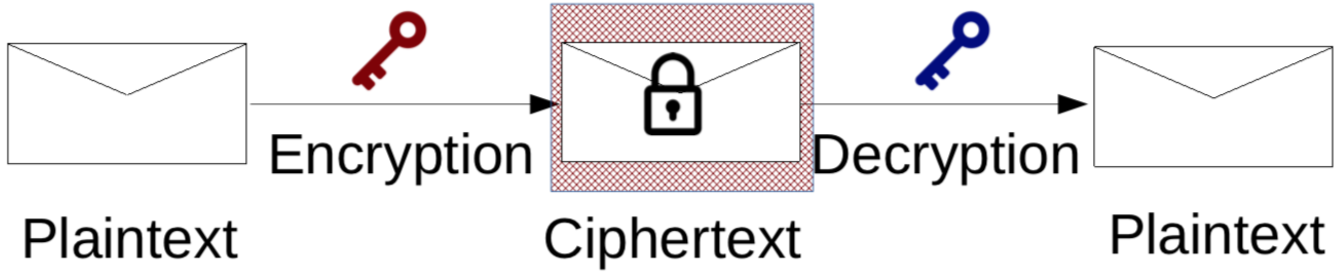
Some slides are written by Mark Stamp.

Public Key Encryption (PKE)

In SKE, locking and opening require the *same key*



What if we want them to require *different keys*?



This is known as *Public Key Encryption*

Public Key Encryption (PKE)

Has two keys for two procedures:

- *Public key* is used for encryption
- *Private key* is used for decryption

Alice generates both keys.

(They are mathematically related.)

Then, Alice publishes her public key:

- *Anyone* can encrypt
- *Only Alice* can decrypt

Anyone can write a message that only Alice can read.

Examples: RSA, ElGamal, ECC

Richard Stallman

American

Richard M
occasiona
movemen
be distribu
to use, stu

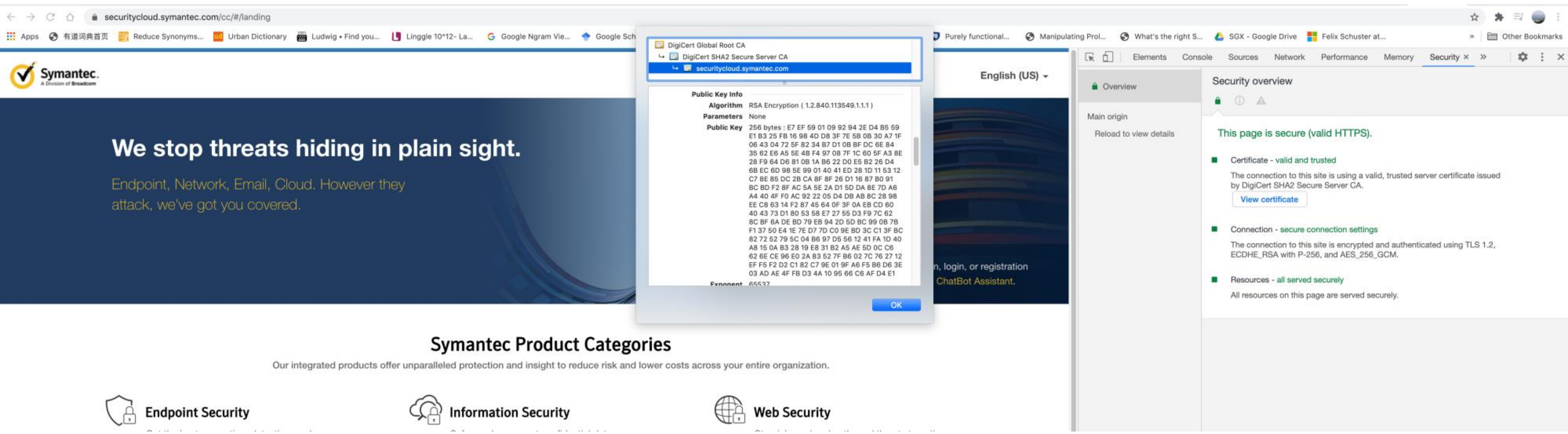
Born: Ma
States

Full nam

← → ⟳ ⌂ stallman.org/rms-pubkey.txt
Show apps
Apps 有道词典首页 Reduce Synonyms... Urban Dictionary Ludwig • Find you... Linggle 10^12- La... Goog

-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.10 (GNU/Linux)

mQINBFHqu6YBEAC/f9aXkt2t+58gGhQiInr3yK/uhQYtmTwxvVVVAEcorRhjMFjC
lPhFsJ7qh0oiCKs7YGh5YSuGTR4YWrF9qS7BzJJNWiu+sFmVPTHiiJ4OoFx4f4dM
9Cl+k3I+orPSuTv5LkMz3omBwl8bt/zPxAeOMV1h6H87zKjTvRdt8K0/XOKuP83d
8pK8gHhIPIBsrQ5YhGImyT8Ni+ffZnjm7IApFKqDJSeMWJ0qJrefwC92i2H/eYcf
LGo/R7VZec9S5Y8xvMejzey9jwPWaQ/Nrxkl2wicg8A3QB4zkqfC61EUGXQr3DE4
fCFv8C5osmiO5kcrMOXZ4GvX4A3CB8O5kXkTsNCS4+Er3Yz/8m7cRCLFze3DjmET
k+rC5zcYdsQ3JiLLwT/5f0btLijEjdv3P9W/LXthV5Sy9L6g9t7RQ5eniO0Sb5f9
fif3geV/NMRUkgZ0nBrwfXgs1iHyixXIV5heke9ncF5IwWdC4pQpkPFq7sFmmqzI
4YgASZmMwHRhjqdFC3wefI8YjgjSesQrgYaYcNM24XZM3OXJKvH9Ky0XUEU+Tfzd
0eefG1inYUO7jbAqLQSBrHB2so9GaPyD87OPsc9kstGjHWKN694Ky+P9sbzNynUO
hJh+XmZd2VUsEqSfvi4amcPVrQK48iP3W42L8eQ6HIw+GUtllHES57ESTQARAQAB
tB5SaWNoYXJkIFN0YWxsbWFuIDxybXNAZ251Lm9yZz6JAjgEEwECACIFAlHqu6YC
GwMGCwkIBwMCBhUIAgkKCwQWAgMBAh4BAheAAAoJECxkZK8qjkwCfE4P/3DywOmg
UOV7eDZhdnWJQ0KC/MOJOPLt1uLoou0A+a5yQYTP4/tSePjfxFuG0x5muaPvY7kI
JwEuILBEZ4dw6VPwtJvO/MLvm5ebHiqjWTw40hNNLiCvokt7rAfZj51FXpIzwhdL
3onk4RLHWzR3XjwIGfAyXImUyUHi1OOrM0oVuLEg1Y1dehyvMKk3OrV2hp3ko3jo
rRpmAT0I5e+CdgH+tghaS+Mrg0LNrnL8o4DJN8U4i8myiLV+8hxc8dGbpcbFJ9wp
UauWYfrKbB9n2Z3foWq7ejHIhJfs1lNmdb2j4oOSAHER94mjk+uxfL+krb20fOZT
9r1uODgYMKzCzcbmWq5IbXHBRBjD6+l2xi5PuwZS1/B0uK9zbhvzv63pwKJrv1bE
PiZpTN6Ck+6pjoFe+TIVqPHnHxwLXVFyIRTVdHnSs4GoO7AuudSeFdItepsUv+bz
qbsn7wut4i43m2raqWf/emGXf8/lF4ClISAF7DvLzWyl91Ep8u6d3WHsWupZfANN
EpyyMHbOanDoHH6P4bxVHko9X36zU5TkWgJNFlYAPubWtrDn1gTQA6HGb9f8cbHL
FxnOsVcS+vE5FgSM2imVT8/JlRBDAf9uI3rYKm/PDRgvu+uHAHPo6Jx3GoXvD1ae
yD49ZO3bMJAUoF69Sv9hG+9VA29B07eEsqVgiEYEEBECAAYFAlHqvNkACgkQYk3F
ZRNepmiLpACgj4o71O0BUB03/I/kpG/n4yRncq4AmwXuXP6yCG9kuLxOwts4BsfF
1Hc6iQIiBBMBAgAMBQJSNiQ2BYMHhh+AAAoJEPBd2uQDcfzl6pQP/Aw8z18OnWq
VEnS25ZnM0nMofDeB+j/PaJJ/OOhru5dRkxRvI1T/BbTyaozrbLEwBawS1cECAZv
b5zUJ3QlSf7wfxiWOW/5u4pzt6Uw2YCPWyi0VU4O8RWUyMhf0r59lciVt7blOj7t
qFY88xdf61zUV1Uvr4SiwxA6uUI3t8XTqnCKGhcQYKphxMBfETmMrOkAZQ0WYihV
IfxzopmPK4s7ItrwpoGKAqR4LA41aprE9icWgmCF+Pqp22Vd5+QFrexGgWxCQ+BS
FxOdadZa/nPsZYhGQU6mIiAJxYj9/FNi2FCgQBNOFnfS5FWai7rIKZnLfH4ZFbTJ
0dgLHtrTUI0+Ab0vglcr9xZbvXU+93FK9MqZs5g7YzY6XiLoeieYMDvSDz0WbJ8q


securitycloud.symantec.com/cc/#/landing
Symantec.
A Division of Broadcom
English (US)
We stop threats hiding in plain sight.
Endpoint, Network, Email, Cloud. However they attack, we've got you covered.
n, login, or registration
ChatBot Assistant.
DigiCert Global Root CA
DigiCert SHA2 Secure Server CA
securitycloud.symantec.com
Public Key Info
Algorithm RSA Encryption (1.2.840.113549.1.1.1)
Parameters None
Public Key 256 bytes
OK
Symantec Product Categories
Our integrated products offer unparalleled protection and insight to reduce risk and lower costs across your entire organization.
Endpoint Security
Information Security
Web Security
Elements
Console
Sources
Network
Performance
Memory
Security
Overview
Main origin
Reload to view details
Security overview
This page is secure (valid HTTPS).
Certificate - valid and trusted
The connection to this site is using a valid, trusted server certificate issued by DigiCert SHA2 Secure Server CA.
View certificate
Connection - secure connection settings
The connection to this site is encrypted and authenticated using TLS 1.2, ECDHE_RSA with P-256, and AES_256_GCM.
Resources - all served securely
All resources on this page are served securely.

Public Key Cryptography

- Generally based on "trap door, one way function"
  - "One way" means easy to compute in one direction, but hard to compute in other direction
  - One example: given p and q, product $N = pq$ easy to compute, but hard to find p and q from N
    - Easy to multiply, but hard to factor
  - "Trap door" is used when creating key pairs

Public Key Cryptography

- Encryption
  - Suppose we **encrypt** M with Bob's public key
  - Bob's private key can **decrypt** C to recover M
- Digital Signature (less obvious)
  - Bob **signs** by "encrypting" with his private key
  - Anyone can **verify** signature by "decrypting" with Bob's public key
  - But only Bob could have signed
  - Like a handwritten signature, but much better…

RSA

RSA

- Invented by **R**ivest, **S**hamir, and **A**dleman
  - RSA is the ***gold standard*** in public key crypto
- Let p and q be two large prime numbers
- Let $N = pq$ be the **modulus**
- Choose e relatively prime to $(p-1)(q-1)$
- Find d such that $ed = 1 \bmod (p-1)(q-1)$
- **Public key** is (N,e)
- **Private key** is d

RSA

- Message M is treated as a number
- To encrypt M we compute
  $C = M^e \bmod N$
- To decrypt ciphertext C, we compute
  $M = C^d \bmod N$
- Recall that e and N are public
- If the attacker can factor $N = pq$, he can use e to easily find d since $ed = 1 \bmod (p-1)(q-1)$
- So, **factoring the modulus breaks RSA**

A proof on the RSA correctness requires some knowledge on number theory; we will give a proof after today's class.

Simple RSA Example

- Example of ***textbook*** RSA
  - Select "large" primes $p = 11$, $q = 3$

- **Public key:**
- **Private key:**

Simple RSA Example

- **Public key:** $(N, e) = (33, 3)$
- **Private key:** $d = 7$
- Suppose message to encrypt is $M = 8$
- Ciphertext C is computed as

  $C =$

- Decrypt C to recover the message M by

  $M =$

RSA Numbers

- RSA lab published a number of semiprimes (N) with 100 to 617 decimal digits
  - numbers with exactly two prime factors
  - As of August 2018, 20 of the 54 listed numbers have been factored.

```
RSA-100 = 15226050279225333605356183781326374297180681149613
          80688657908494580122963258952897654000350692006139
RSA-100 = 37975227936943673922808872755445627854565536638199
        × 40094690950920881030683735292761468389214899724061
```

RSA-100: the easiest challenge with 330 bits.

```
RSA-2048 = 2519590847565789349402718324004839857142928212620403202777713783604366202070
           7595556264018525880784406918290641249515082189298559149176184502808489120072
           8449926873928072877767359714183472702618963750149718246911650776133798590957
           0009733045974880842840179742910064245869181719511874612151517265463228221686
           9987549182422433637259085141865462043576798423387184774447920739934236584823
           8242811981638150106748104516603773060562016196762561338441436038339044149526
           3443219011465754445417842402092461651572335077870774981712577246796292638635
           6373289912154831438167899885040445364023527381951378636564391212010397122822
           120720357
```

RSA-2048: no one can break.

RSA Key Collisions

- Chances are very low.
  - An informal explanation.

But what about the efficiency?

What if the key are very large? (over 2000 bits)

```
>>> 17 ** 2222222222222222222222222222222222222222222
^CTraceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyboardInterrupt
>>> 17 ** 222222222222222222222222222222222
^CTraceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyboardInterrupt
>>> 17 ** 22222222222222
^CTraceback (most recent call last):
  File "<stdin>", line 1, in <module>
KeyboardInterrupt
>>> 17 ** 222222 <----- a very large number
```

4294967295 <---- 32-bit integer

More Efficient RSA

- What if the key are very large? (over 2000 bits)
- Can we still use modular exponentiation?
  - $5^{20} = 95367431640625 = 25 \bmod 35$ → very inefficient
  - Mod each step, without waiting until it gets large!
- A better way: **repeated squaring**
  - $20 = 1 * 2 * 5 * 2$ <-- simplified!!
  - $5^1 = 5 \bmod 35$
  - $5^2 = (5^1)^2 = 5^2 = 25 \bmod 35$
  - $5^5 = (5^2)^2 \cdot 5^1 = 25^2 \cdot 5 = 3125 = 10 \bmod 35$
  - $5^{10} = (5^5)^2 = 10^2 = 100 = 30 \bmod 35$
  - $5^{20} = (5^{10})^2 = 30^2 = 900 = 25 \bmod 35$
- No huge numbers and it's efficient!
  - Can precompute a table of 5^x (where x is some common numbers)

Detailed algorithm is not required to remember. We will come back to this later this semester to discuss some attacks on RSA.

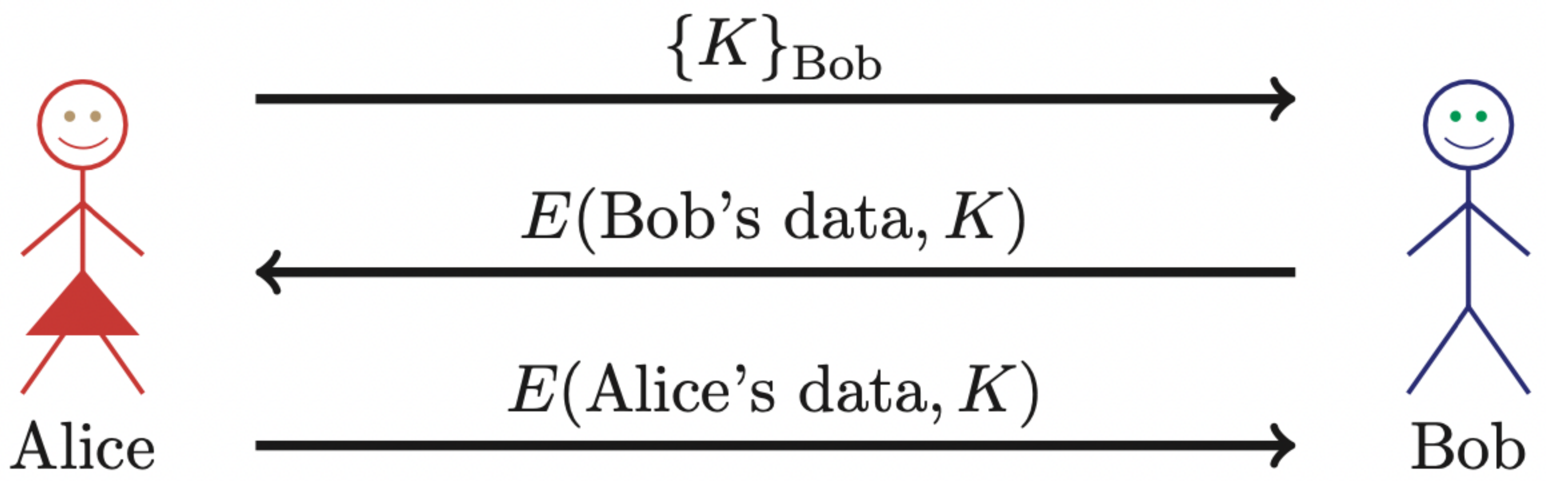
RSA Efficiency and Limits

- Encrypt message of limited size
  - Remember we need to do a **mod** operation..
  - In practice, for 2048 bits RSA key, it could be 245 bytes plaintext
- RSA + symmetric key encryption (**hybrid cryptosystem**)
  - That’s what we are doing (SSL/OpenPGP).
  - RSA isn't designed as a full-speed data transport cipher
- Or cut messages into chunks of 245 bytes?
  - Bad idea (size increase; and still, it’s generally slow)

Details of repeated squaring are not required. We will come back to this later when discussing attacks.

Real World Confidentiality

- **Hybrid cryptosystem**
  - Public key crypto to establish a key
  - Symmetric key crypto to encrypt data…



❑ Can Bob be sure he's talking to Alice?

Public Key Crypto vs. Symmetric Key Crypto

	PKE	SKE
Key	Two: public/private	One: secret
Key setup	Share public key	Need safe channel
Encrypt	Anyone	Both participants
Decrypt	Only key generator	Both participants
Efficiency	Costly to encrypt/decrypt	Cheap

We can combine PKE and SKE to cover their weakness!

- But still need certificates from a trusted third party. ← talk later.

Quantum Computers and Public Key Crypto

- Recall that quantum computing ***not*** a serious threat to symmetric ciphers
- But, QC is a **BIG** threat to public key
- Shor's factoring algorithm (1994)
  - Most famous quantum algorithm
- Let n be number of bits in N, then…
  - Work factor of $n^2\log_2(n)\log_2(\log_2(n))$

Quantum Computers and Public Key Crypto

- Shor's algorithm much faster than best classic factoring algorithm
  - Number field sieve is best classic alg.
- For a 2048-bit modulus, work factor…
  - Number field sieve equivalent to exhaustive search for 125-bit key
  - Shor's algorithm equivalent to exhaustive search for 30-bit key

Quantum Computers and Public Key Crypto

- Bottom line?
- QC will make RSA obsolete
- Post-quantum cryptography?
  - Symmetric ciphers will be OK
  - Most popular public key algorithms will fail (RSA, Diffie-Hellman, …)
- But there exist public key algorithms that are secure against QC
  - For example, NTRU

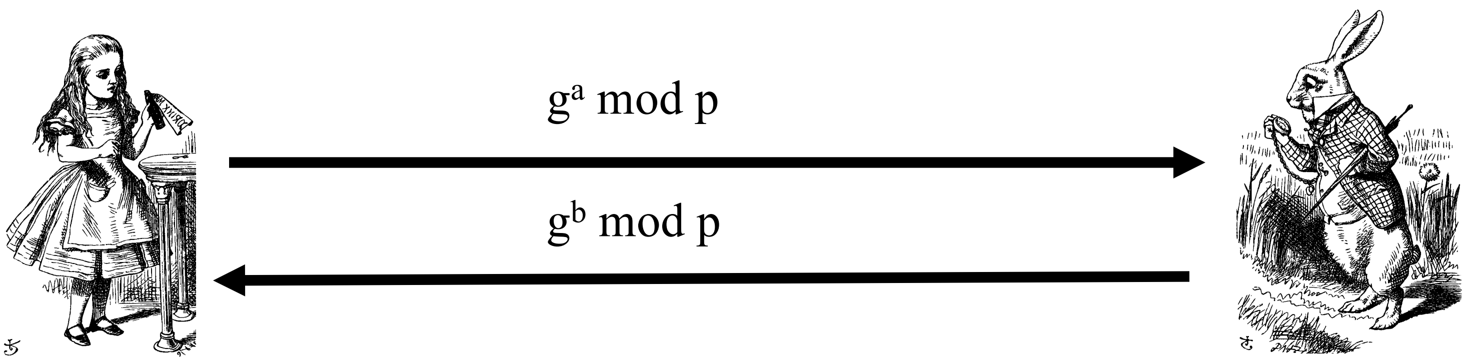
Diffie-Hellman

Diffie-Hellman Key Exchange

- A "key exchange" algorithm
  - Used to establish a shared symmetric key
  - ***Not*** for encrypting or signing
- Based on **discrete log** problem
  - **Given:** g, p, and $g^k \bmod p$
  - **Find:** exponent k
  - Very hard problem…

Diffie-Hellman

- **Public:** g and p
- **Private:** Alice's exponent a, Bob's exponent b



Alice, a

Bob, b

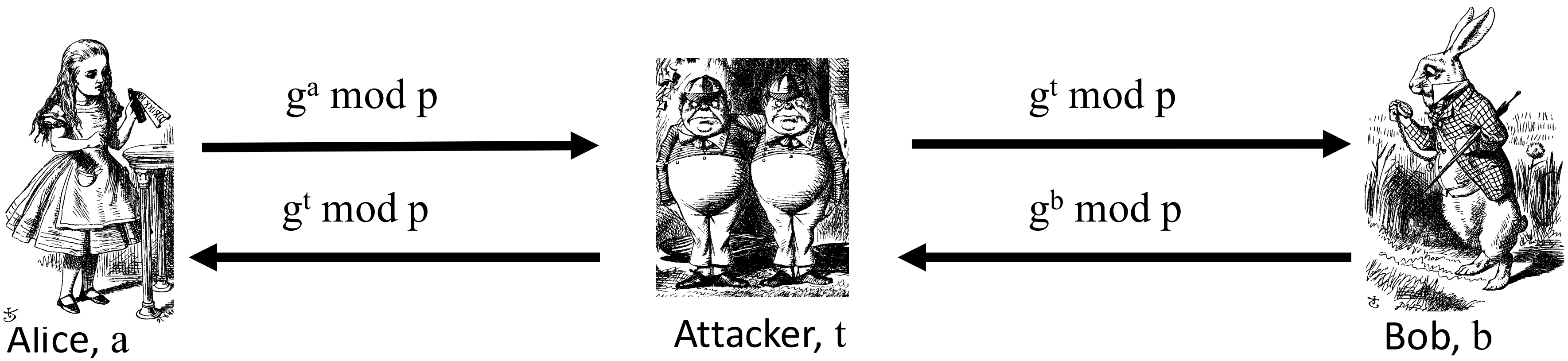
- Alice computes $(g^b \bmod p)^a = g^{ba} \bmod p = g^{ab} \bmod p$
- Bob computes $(g^a \bmod p)^b = g^{ab} \bmod p$
- They can use $K = g^{ab} \bmod p$ as symmetric key

Diffie-Hellman

- Suppose Bob and Alice use Diffie-Hellman to determine symmetric key $K = g^{ab} \bmod p$
- The attacker can see $g^a \bmod p$ and $g^b \bmod p$
  - But… $g^a g^b \bmod p = g^{a+b} \bmod p \neq g^{ab} \bmod p$
- If attacker can find a or b, she gets K
- If attacker can solve **discrete log** problem, she can find a or b

Diffie-Hellman

- Subject to man-in-the-middle (MiM) attack



- Trudy shares secret g^{at} mod p with Alice
- Trudy shares secret g^{bt} mod p with Bob
- Alice and Bob don't know attacker is MiM

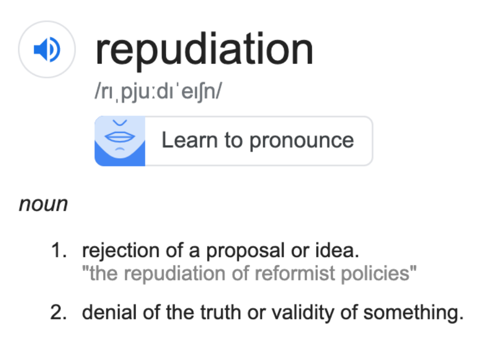
Diffie-Hellman

- How to prevent MiM attack?
  - Alice and bob do not share any thing else..
- At this point, DH may look pointless…
  - …but it’s not (more on this later) → need authentication
- You **MUST** be aware of MiM attack on Diffie-Hellman

Signature with RSA

Uses for Public Key Crypto

- Confidentiality
  - Transmitting data over insecure channel
  - Secure storage on insecure media
- Authentication protocols (later)
- Digital signature
  - Use private RSA key to “encrypt” message, why?
  - Provides **integrity** and **non-repudiation**
  - No non-repudiation with symmetric keys



Repudiation (Non-non-repudiation)

- Alice orders 100 shares of stock from Bob
- Alice computes **MAC** using symmetric key
- Stock drops, Alice claims she did ***not*** order
- Can Bob prove that Alice placed the order?
- **No!** Bob also knows the symmetric key, so he could have forged the **MAC**
- **Problem:** Bob knows Alice placed the order, but he can't prove it

Non-repudiation

- Alice orders 100 shares of stock from Bob
- Alice **signs** order with her private key
- Stock drops, Alice claims she did not order
- Can Bob prove that Alice placed the order?
- **Yes!** Alice’s private key used to sign the order — only Alice knows her private key
- This assumes Alice’s private key has not been lost/stolen

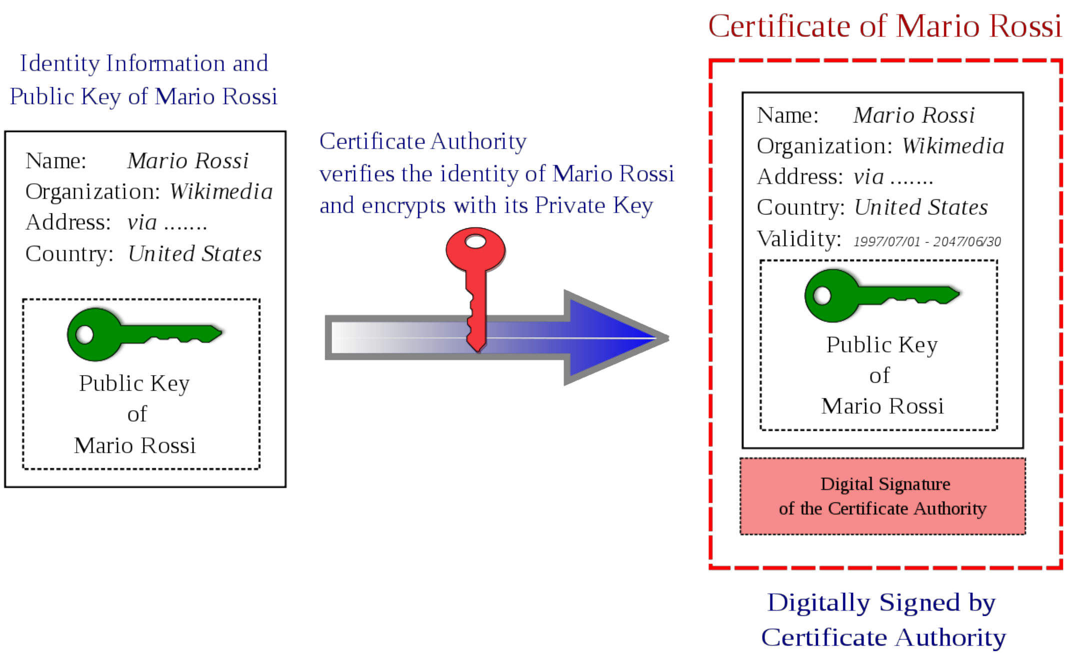
Public Key Infrastructure

Public Key Certificate

- How can the recipient know with certainty the sender's public key? (to validate a digital signature)
- How can the sender know with certainty the recipient's public key? (to send an encrypted message)
- Use a trusted third party to authenticate that the public key belongs to A → **Certification Authority (CA)**
- For each user A, **CA** creates a message containing A's name and public key (**Digital Certificate**).

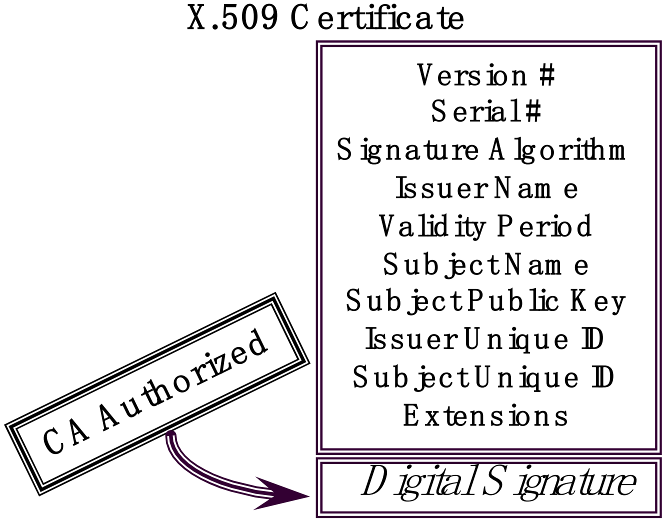
Public Key Certificate

- **Digital certificate** contains name of user and user's public key (possibly other info too)
- It is ***signed*** by the issuer, a ***Certificate Authority*** (CA)
- Signature on certificate is verified using CA's public key

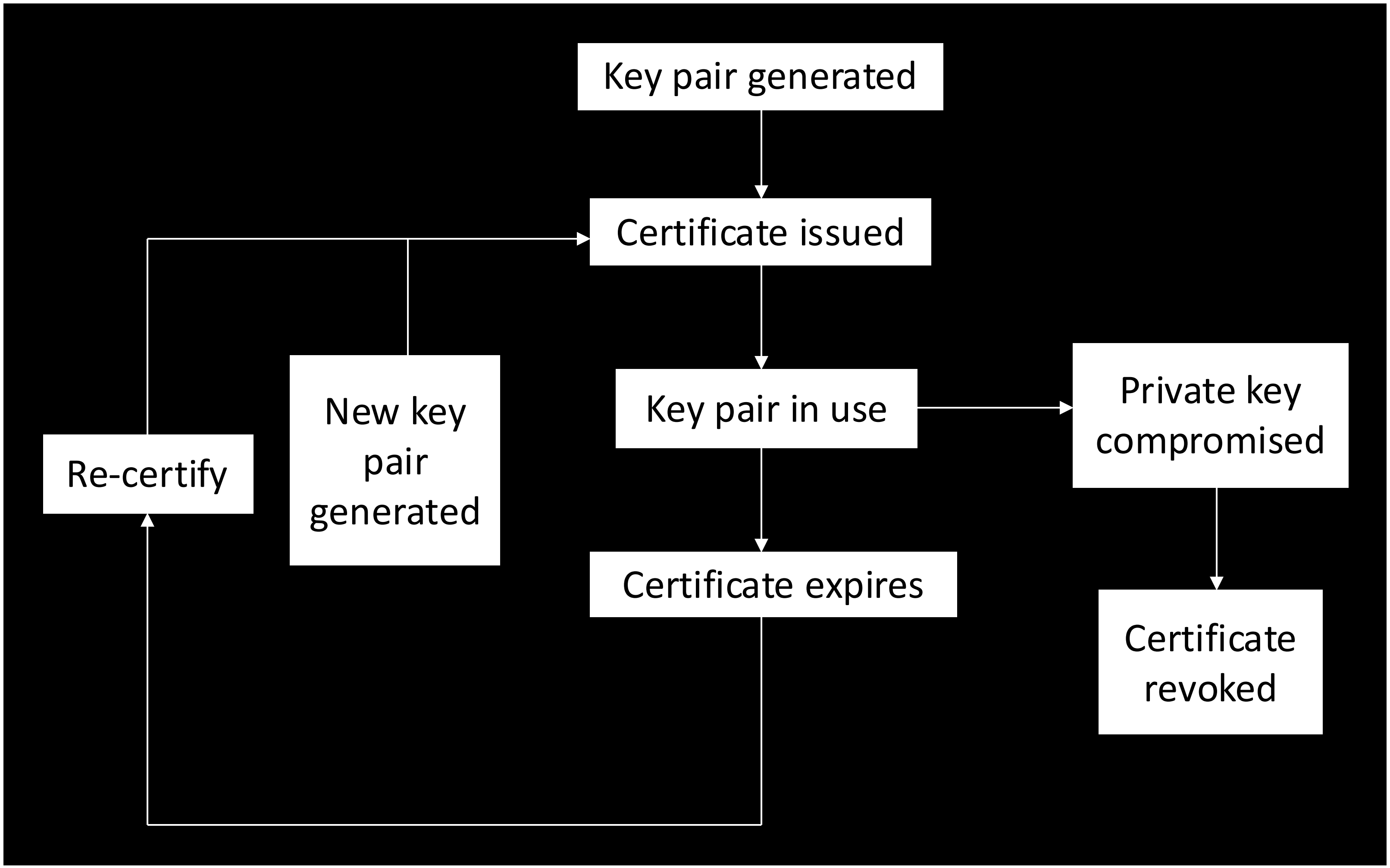


Certificate Authority (CA)

- Certificate authority (CA) is a trusted 3rd party — creates and signs certificates
- Verify signature to verify **integrity**
- A common format for certificates is X.509



Certificate Life Cycle



Why do I trust a CA? And Why do a CA trust me?

- “Root CA” stored in your OS/browser/application since installation time
  - Updated as part of normal secured (OS/Application) updates
- Verification of identity by certification authority
  - the identity of a user is verified by real-world measures, not cryptography matters.

PKI

- Public Key Infrastructure (PKI): the stuff needed to securely use public key crypto
  - Key generation and management
  - Certificate authority (CA) or authorities
  - Certificate revocation lists (CRLs), etc.
- No general standard for PKI
- 3 generic "trust models"
  - Monopoly model
  - A few trusted CAs (most common approach)
  - Everyone is a CA…

Hash Functions

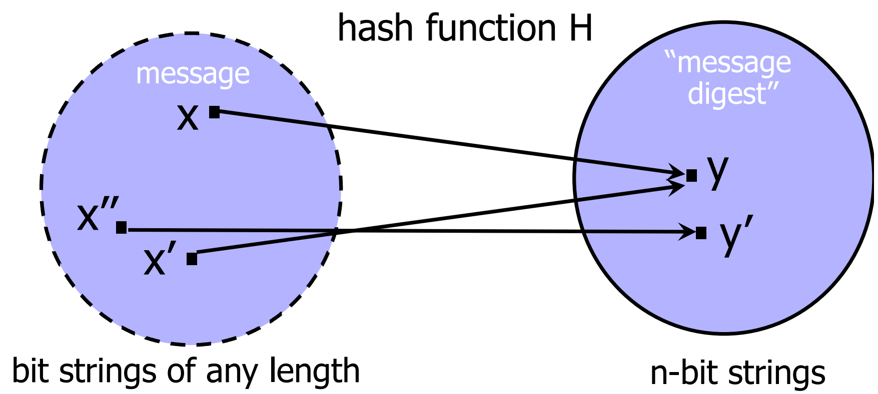
Hash Function Motivation

- Some terms
  - **Sign** message M with Alice's **private key:** $[M]_{\text{Alice}}$
  - **Encrypt** message M with Alice's **public key:** $\{M\}_{\text{Alice}}$
- Suppose Alice signs M
  - Alice sends M and $S = [M]_{\text{Alice}}$ to Bob
  - Bob verifies that $M = \{S\}_{\text{Alice}}$
- If M is big, $[M]_{\text{Alice}}$ costly to ***compute*** & ***send***
- Suppose instead, Alice signs $h(M)$, where $h(M)$ is a much smaller "fingerprint" of M
  - Alice sends M and $S = [h(M)]_{\text{Alice}}$ to Bob
  - Bob verifies that $h(M) = \{S\}_{\text{Alice}}$

Hash Function Motivation

- So, Alice signs $h(M)$
  - That is, Alice computes $S = [h(M)]_{Alice}$
  - Alice then sends (M, S) to Bob
  - Bob verifies that $h(M) = \{S\}_{Alice}$
- What properties must $h(M)$ satisfy?
  - Suppose attacker finds M' so that $h(M) = h(M')$
  - Then attacker can replace (M, S) with (M', S)
- Does Bob detect this tampering?
  - No, since $h(M') = h(M) = \{S\}_{Alice}$

Crypto Hash Function



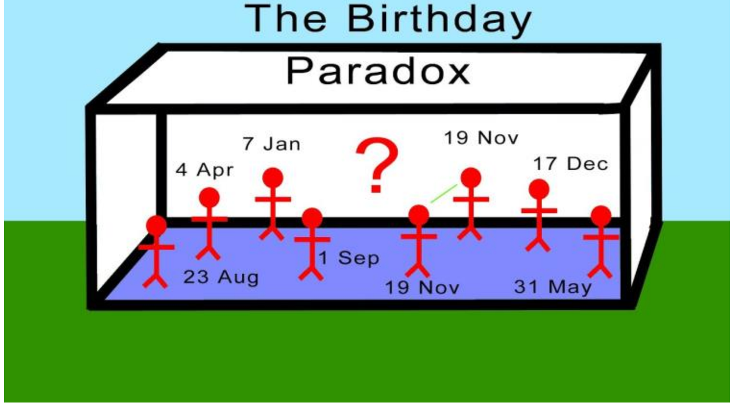
- Crypto hash function $H(x)$ shall provide
  - **Compression** — output length is small
  - **Efficiency** — $H(x)$ easy to compute for any x
  - **One-way** — given a value y it is infeasible to find an x such that $H(x) = y$
  - **Weak collision resistance** — given x and $H(x)$, infeasible to find $y \neq x$ such that $H(y) = H(x)$ ➔ pre-birthday problem
  - **Strong collision resistance** — infeasible to find any $y \neq x$ such that $H(y) = H(x)$ ➔ birthday problem

Pre-Birthday Problem

- Suppose N people in a room
- How large must N be before the probability someone has same birthday as me is $\geq 1/2$?
  - Solve: $1/2 = 1 - (364/365)^N$ for N
  - We find $N = 253$

Birthday Problem

- How many people must be in a room before probability is ≥ 1/2 that any two (or more) have same birthday?
  - $1 - 365/365 \cdot 364/365 \cdots (365-N+1)/365$
  - Set equal to 1/2 and solve: **N = 23**
- "Should be" about sqrt(365) since we compare all **pairs** x and y
  - And there are 365 possible birthdays



Of Hashes and Birthdays

- If $h(x)$ is N bits, then 2^N different hash values are possible
- **ESTIMATION:** so, if you hash about $\mathrm{sqrt}(2^N) = 2^{N/2}$ values then you expect to find a collision
- **Implication?** "Exhaustive search" attack…
  - Secure N-bit hash requires $2^{N/2}$ work to "break"
  - Secure N bit symmetric key require about 2^N work to break
  - If N = 128, how many works need to be done to break?

We will explain how this "ESTIMATION" is achieved with some notes after this lecture.

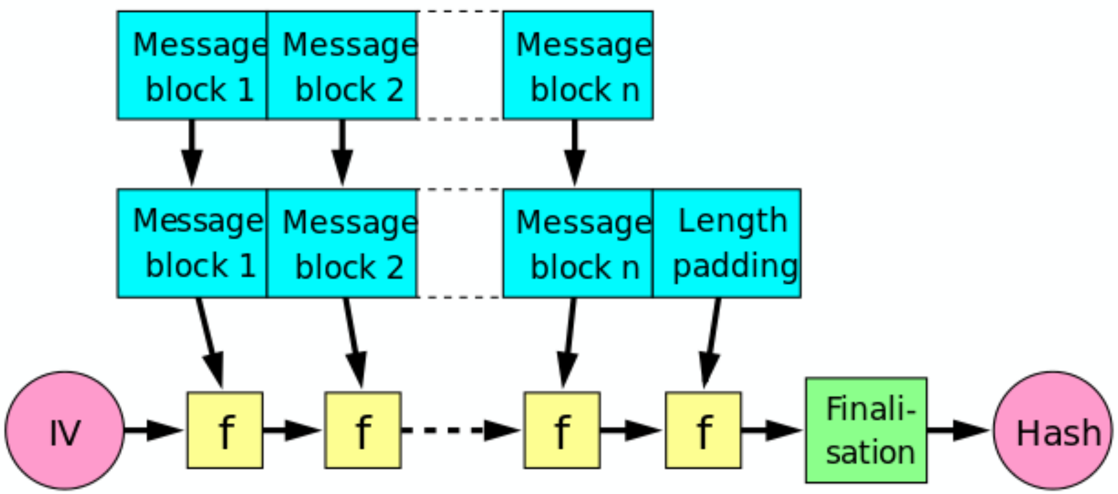
Popular Crypto Hashes

- **MD5** (Invented by Rivest)
  - 128-bit output
  - MD5 collisions easy to find, so it's broken
- **SHA-1** — A U.S. government standard, inner workings similar with MD5
  - 160-bit output
  - Well, but also crack-able since 2005…
- Many other hashes
  - SHA-224; SHA-364; SHA-512
- Hashes work by hashing message in blocks
  - Very very similar to block cipher (e.g., DES)
  - There is no **key** involved.

Internals of a Hash Function

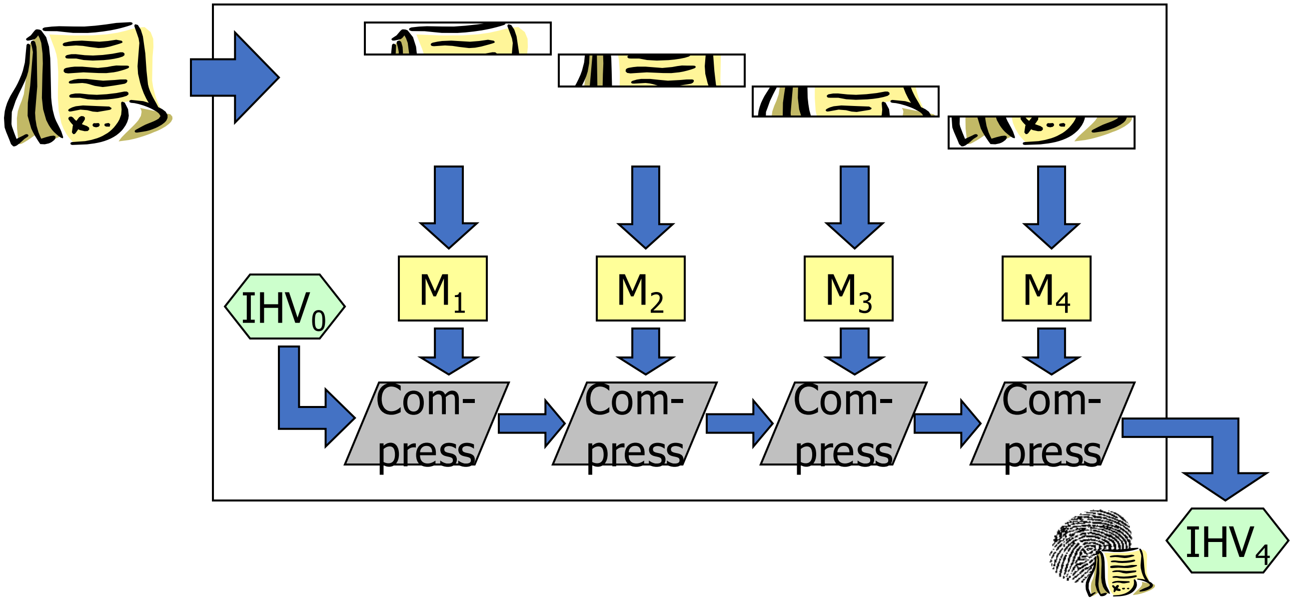
Merkle-Damgard construction:

- A fixed-size "compression function".
- Each iteration mixes an input block with the previous output.



MD5

- Iterative design using compression function



avalanche effect

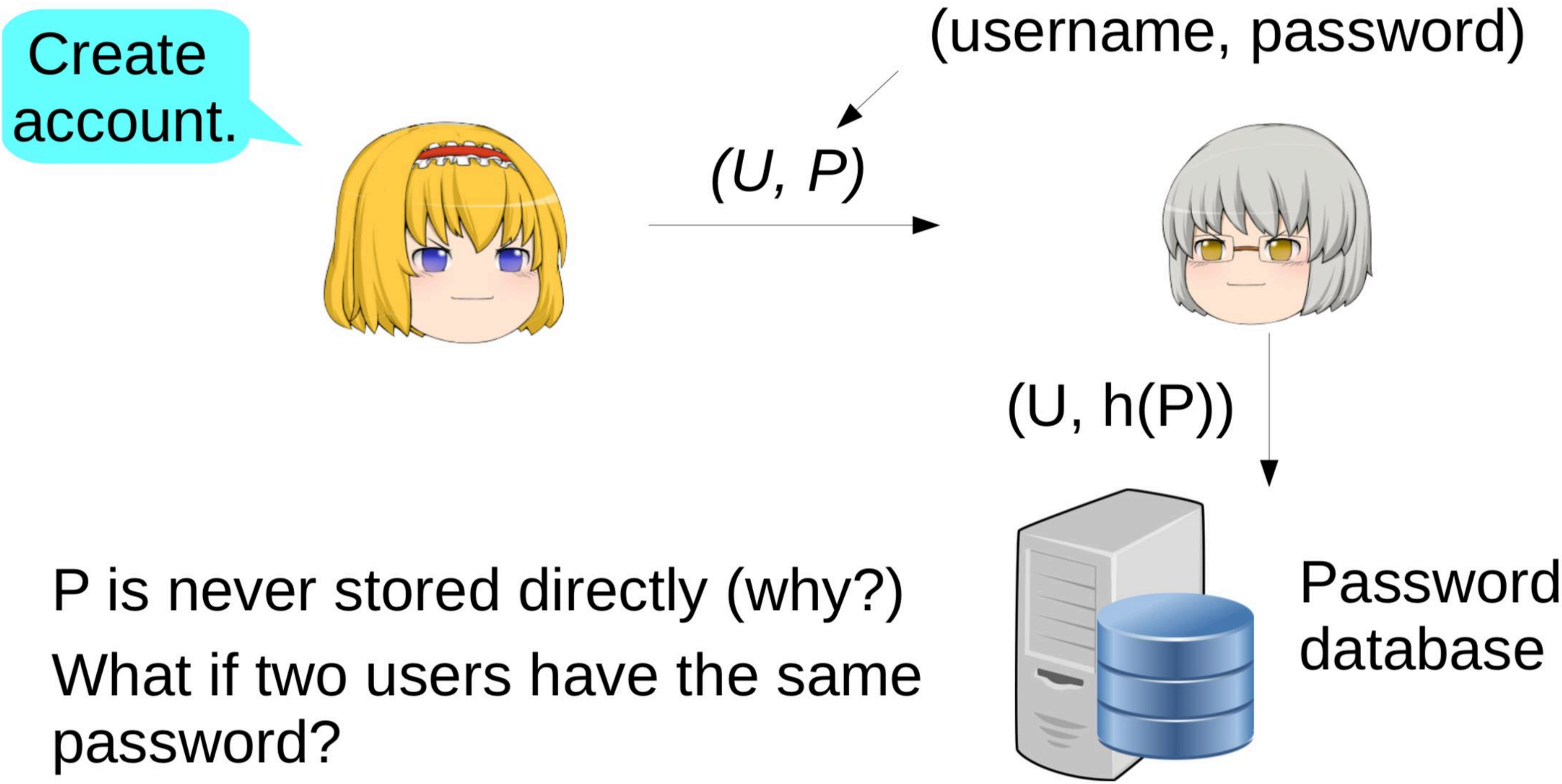
MD5(“The quick brown fox jumps over the lazy dog”) = 9e107d9d372bb6826bd81d3542a419d6

MD5("The quick brown fox jumps over the lazy dog.") = e4d909c290d0fb1ca068ffaddf22cbd0

Hash Uses

- Authentication
- Message integrity
- Message fingerprint
- Data corruption detection
- Digital signature efficiency
- And many other smart usages…

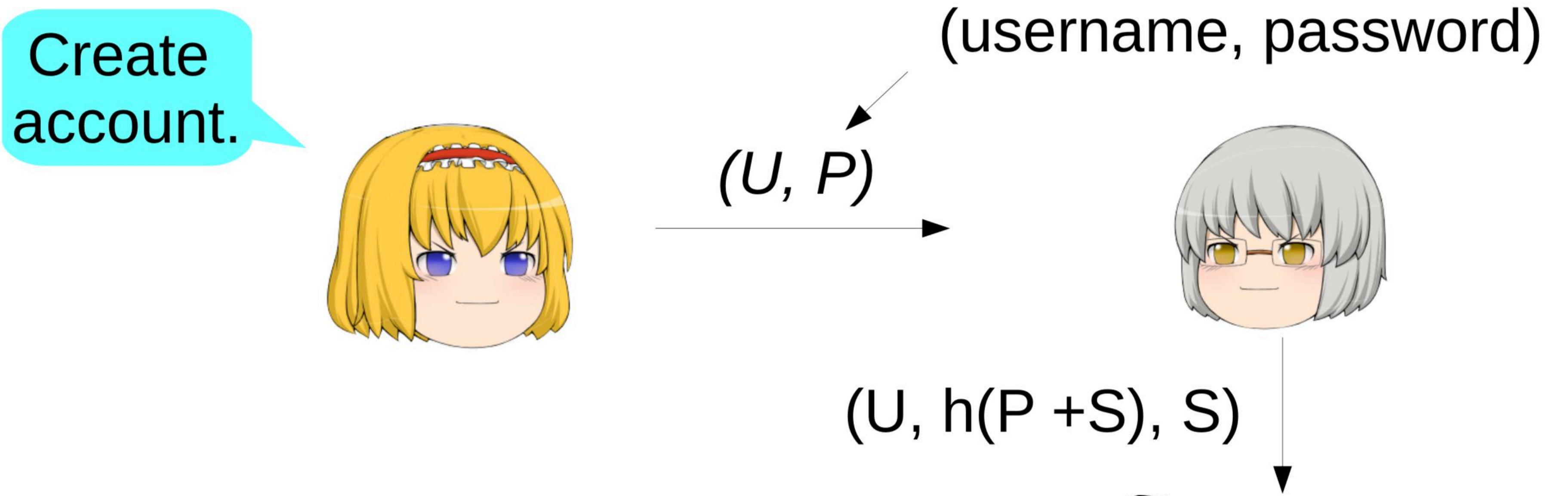
How companies store your password?



P is never stored directly (why?)

What if two users have the same password?

Password Hashing



P is never stored directly (why?)

What if two users have the same password?

Add a random *Salt* to the password

Using hashes as authenticators

- Prof. Alice wants to cancel today's class, by communicating to the students via Trudy.

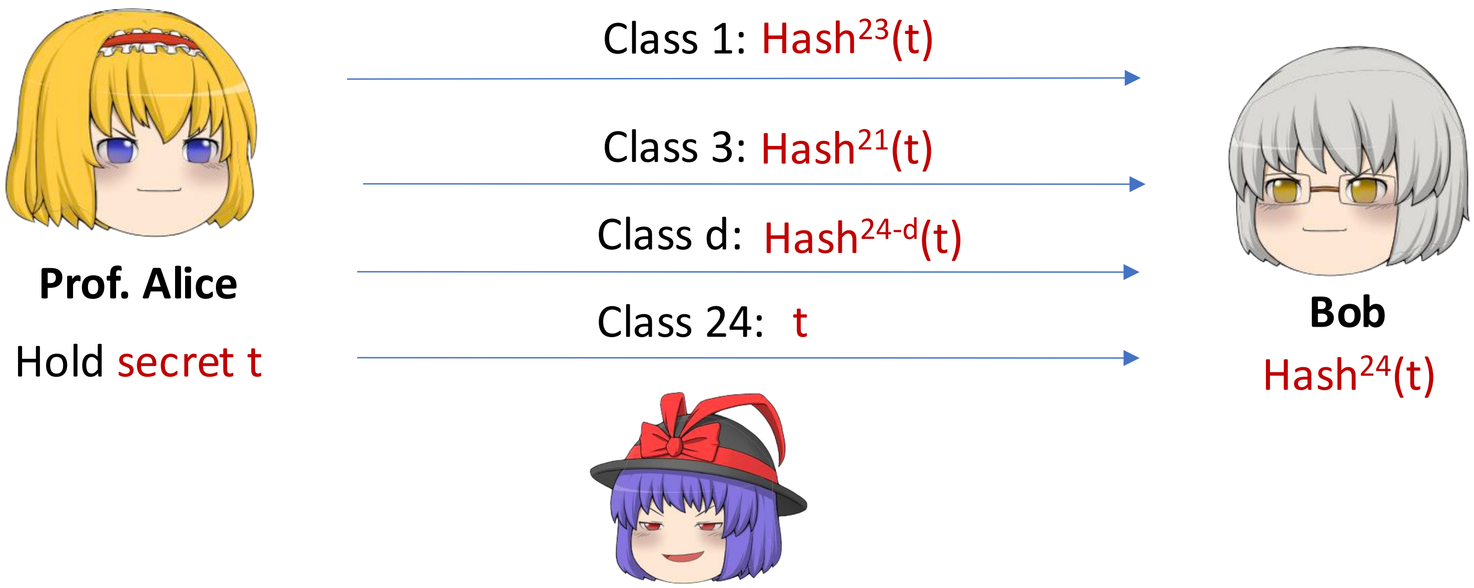
Threat model: Trudy tries to make Bob NOT attend lectures.

Why is this protocol (in)-secure?

- t acts as an authenticated value (authenticator) because Trudy could not have produced t without inverting hash()
- But what if the the Trudy reuse the "t" in the future?
  - then Bob keeps NOT attending the lectures. Attack succeed (and you lost your quiz points)!

Hash Chains

- Suppose we have in total 24 classes.



Why is this protocol secure?

- Trudy cannot infer Hash^{24-d}, by having Hash$^{24-(d-1)}$ on hand.
    - Chain of hash values are ordered authenticators

(Simplified) Token Devices

- A one-time password system that essentially uses a hash chain as authenticators.
  - For seed (S) and chain length (l), current iteration (i)
  - Token encodes S in the hardware (firmware)

$$\mathrm{pw}_i = h^{l-i}(S)$$

  - Device display shows password for iteration i
- Your token display at some time does not disclose information in the future.

Hashing for Spam Reduction

- Spam reduction
- Before accept email, want proof that sender had to "work" to create email
  - Here, "work" == CPU cycles
  - "Proof-of-work" in Blockchain → talk more later.
- Goal is to limit the amount of email that can be sent
  - This approach will not eliminate spam
  - Instead, make spam more costly to send

Spam Reduction

- Let M = complete email message

  $\mathbf{R}$ = value to be determined

  T = current time
- Sender must determine $\mathbf{R}$ so that

  $h(M,\mathbf{R},T) = (00\ldots0,X)$, that is,

  initial N bits of hash value are **all zero**
- Sender then sends $(M,\mathbf{R},T)$
- Recipient accepts email, provided that…

  $h(M,\mathbf{R},T)$ begins with N zeros

Spam Reduction

- Sender: $h(M,R,T)$ begins with N zeros
- Recipient: verify that $h(M,R,T)$ begins with N zeros
- **Work for sender:** on average **2^N hashes**
- **Work for recipient:** always **1 hash**
- Sender's work increases exponentially in N
- Small work for recipient, regardless of N
- Choose N so that…
  - Work acceptable for normal amounts of email
  - Work is too high for spammers