TU Graz

SCIENCE
PASSION
TECHNOLOGY

Cryptography 4:

Protocols and Applications

Maria Eichlseder

Information Security – WT 2023/24

You Are Here

Crypto 1	Crypto 2	Crypto 3	Crypto 4
Symmetric Authentication	Symmetric Encryption	Asymmetric Cryptography	Protocols and Applications
Integrity	Confidentiality	Establishing communication	Theory meets Practice
Hash functions	AEAD (Auth. Encryption)	Key exchange	Protocols
MACs (Message Authentication)	Symmetric primitives	Signatures	Applications
		Asymmetric primitives	Discussion

Recap of Last Week: Asymmetric Primitives = Hard Problems

Discrete Logarithm Problem (DLP)

Given a prime number p, a generator $g \in \mathbb{Z}_p^*$, and an element $y \in \mathbb{Z}_p^*$, find x such that $\underbrace{g \cdot g \cdots g}_{x \text{ times}} = g^x \equiv y \pmod{p}$.

…and the closely related Diffie–Hellman Problem (DHP)

Integer Factorization Problem (IFP)

Given $n \in \mathbb{N}$, find primes p_i and exponents $e_i \in \mathbb{N}$ such that $n = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_k^{e_k}$

…and the closely related RSA Problem (RSAP)

Outline

Protocols

- TLS and Forward Secrecy
- Certificates
- Protocols for Privacy

Crypto Security

- Keys and Randomness
- Quantum Computer Attacks
- Crypto Failures

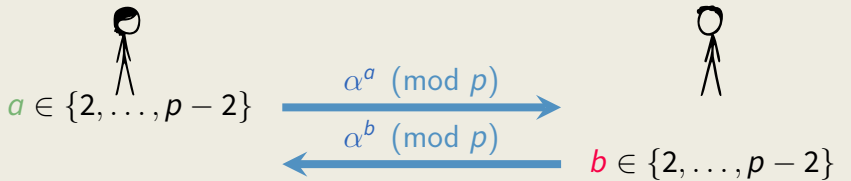
Protocols

TLS & more

Recap & Discussion: Diffie–Hellman Key Exchange

Diffie–Hellman Key Exchange

Choose a large prime p and a generator α of $\mathbb{Z}_p^*$ (public system parameters).



- No authenticity: Alice has no assurance who she is exchanging keys with.
- No forward secrecy: If Alice's private key a leaks, all her past communication is compromised.

Example applications with such “Static Asymmetric Crypto”

Static approach =
Key agreement using long-term asymmetric keypair (DH or RSA encryption), then use this key for symmetric encryption+MAC

- OpenPGP: Authentication by hand based on public-key fingerprint
- S/MIME: Authentication of public keys with certificates
- iMessage: Authentication by long term public key served by Apple

Solution: Ephemeral Diffie–Hellman (DHE)

Solution for both problems: DHE = Diffie–Hellman with Ephemeral keys

- Alice and Bob each use 2 public-key keypairs:
  - Long-term keypair for a signature scheme
  - Short-term (ephemeral, single-use) keypair for DH key exchange

- After executing DH and deriving the symmetric session key $K_{AB} = \alpha^{a \cdot b}$, they
  - send each other a signature over the exchange for authentication
  - send each other the MAC of the exchange using K_{AB} to confirm they know K_{AB}
  - throw away their ephemeral keys

Transport Layer Security (TLS)

Byte-by-byte example here

The TLS protocol is the successor of SSL and responsible for the “`s`” in `https`.

It works in 3 phases:

1. Handshake: Key exchange
   - (Ephemeral) Diffie-Hellman
2. Handshake: Authentication
   - Signature + Certificate
3. Application data
   - Authenticated Encryption (AEAD)

TLS v1.3 Phase 1 – Key Exchange

- $\text{ciphers}_{\square,\equiv}$: list of preferred symmetric ciphersuites
- $r_{\square}, r_{\equiv}$: some fresh randomness (for nonces)
- $k_{\square}, k_{\equiv}$: ephemeral (temporary) DH private keys

TLS v1.3 Phase 2 – Authentication

- Cert: X.509 certificate for server's Sign key
- Sign(τ): Signature over transcript of all handshake (HS) messages so far
- HMAC$_{,;\mathrm{HS}}(\tau', \tau'')$: MAC over transcript, key derived from $\alpha^{k\cdot k}$
- All messages protected by AEAD, key derived from $\alpha^{k\cdot k}$

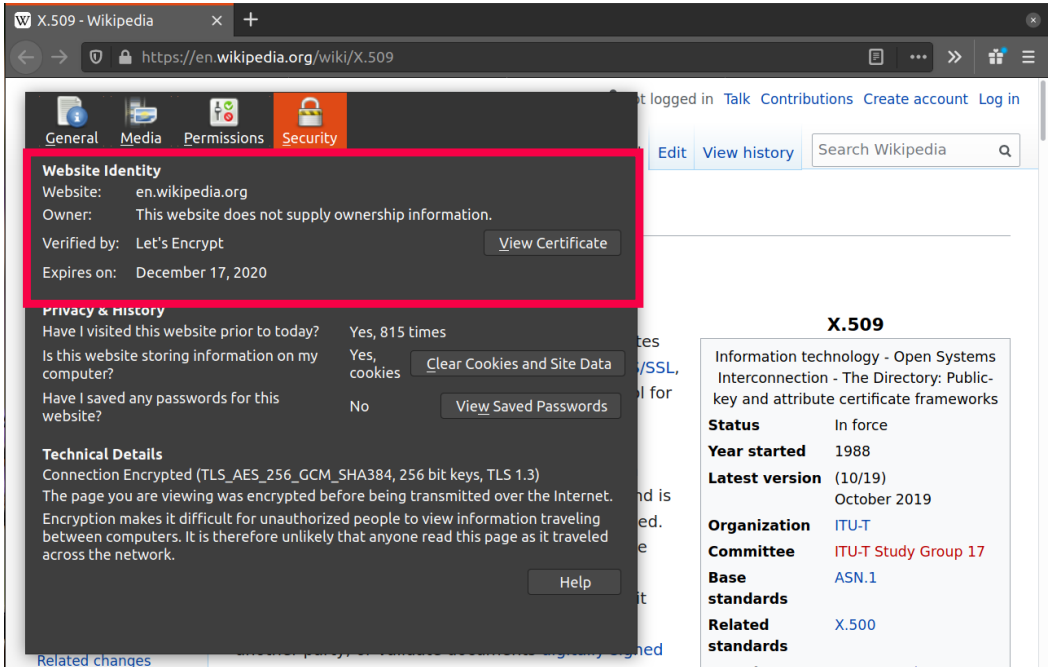
TLS v1.3 Phase 3 – Application Data

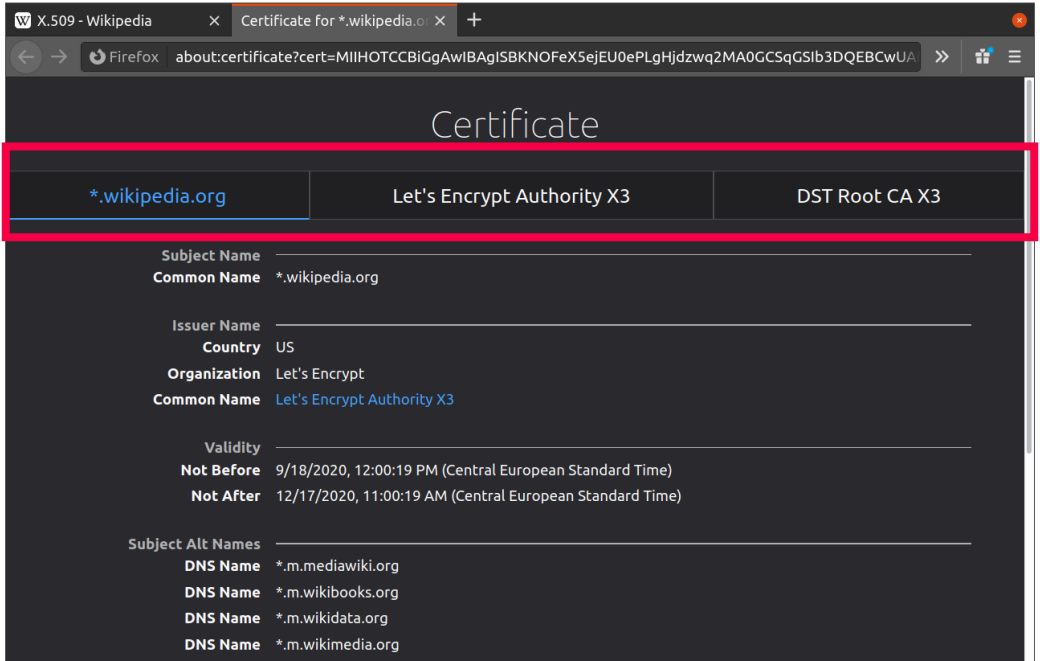
- All messages protected by AEAD
- All symmetric keys derived from $\alpha^{k_{\text{💻}} \cdot k_{\text{🖥}}}$ with a key derivation function (HKDF)

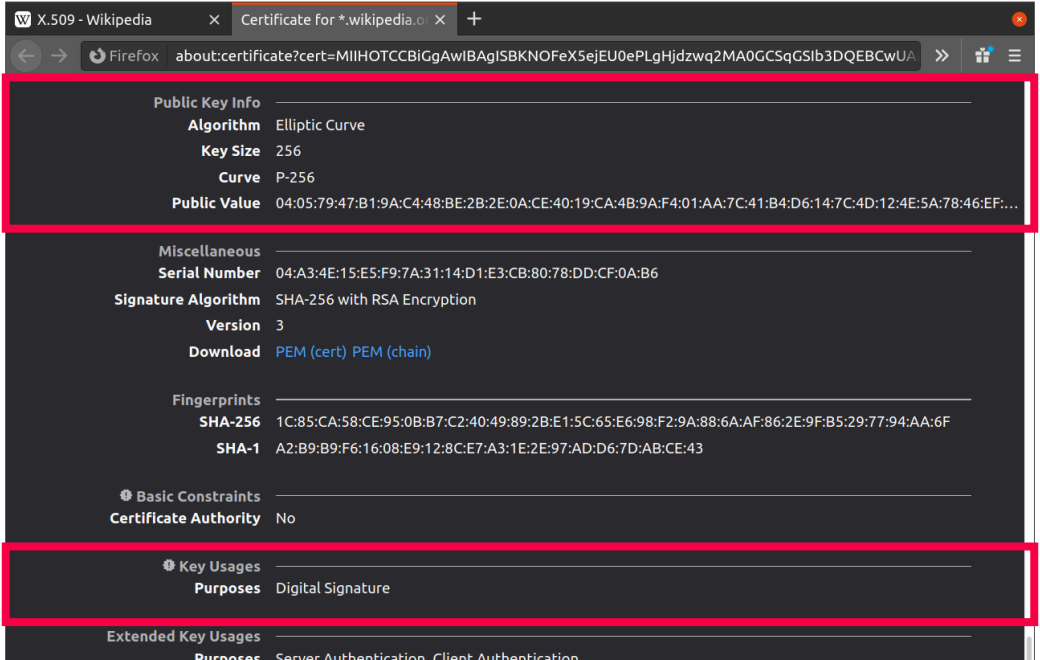
X.509 Certificates

How to ensure that public keys of signatures are authentic?

- A certificate ties a public key to an identity
- X.509 is a standard for such certificates, which contain:
  - A public key
  - Identity information of the owner (e.g., hostname, person's name)
  - Validity period and other constraints
  - Signature from a certificate authority (CA), the "Issuer"
- Your browser, OS, etc. ships with a list of trusted CAs


X.509 - Wikipedia
https://en.wikipedia.org/wiki/X.509
t logged in Talk Contributions Create account Log in
Edit View history
Search Wikipedia
General Media Permissions Security
Website Identity
Website: en.wikipedia.org
Owner: This website does not supply ownership information.
Verified by: Let's Encrypt
View Certificate
Expires on: December 17, 2020
Privacy & History
Have I visited this website prior to today? Yes, 815 times
Is this website storing information on my computer? Yes, cookies
Clear Cookies and Site Data
Have I saved any passwords for this website? No
View Saved Passwords
Technical Details
Connection Encrypted (TLS_AES_256_GCM_SHA384, 256 bit keys, TLS 1.3)
The page you are viewing was encrypted before being transmitted over the Internet.
Encryption makes it difficult for unauthorized people to view information traveling between computers. It is therefore unlikely that anyone read this page as it traveled across the network.
Help
X.509
Information technology - Open Systems Interconnection - The Directory: Public-key and attribute certificate frameworks
Status In force
Year started 1988
Latest version (10/19) October 2019
Organization ITU-T
Committee ITU-T Study Group 17
Base standards ASN.1
Related standards X.500
Related changes


X.509 - Wikipedia
Certificate for *.wikipedia.o
Firefox
about:certificate?cert=MIIHOTCCBiGgAwIBAgISBKNOFeX5ejEU0ePLgHjdzwq2MA0GCSqGSIb3DQEBCwUA
Certificate
*.wikipedia.org
Let's Encrypt Authority X3
DST Root CA X3
Subject Name
Common Name *.wikipedia.org
Issuer Name
Country US
Organization Let's Encrypt
Common Name Let's Encrypt Authority X3
Validity
Not Before 9/18/2020, 12:00:19 PM (Central European Standard Time)
Not After 12/17/2020, 11:00:19 AM (Central European Standard Time)
Subject Alt Names
DNS Name *.m.mediawiki.org
DNS Name *.m.wikibooks.org
DNS Name *.m.wikidata.org
DNS Name *.m.wikimedia.org


X.509 - Wikipedia
Certificate for *.wikipedia.o
Firefox
about:certificate?cert=MIIHOTCCBiGgAwIBAgISBKNOFeX5ejEU0ePLgHjdzwq2MA0GCSqGSIb3DQEBCwUA
Public Key Info
Algorithm Elliptic Curve
Key Size 256
Curve P-256
Public Value 04:05:79:47:B1:9A:C4:48:BE:2B:2E:0A:CE:40:19:CA:4B:9A:F4:01:AA:7C:41:B4:D6:14:7C:4D:12:4E:5A:78:46:EF:...
Miscellaneous
Serial Number 04:A3:4E:15:E5:F9:7A:31:14:D1:E3:CB:80:78:DD:CF:0A:B6
Signature Algorithm SHA-256 with RSA Encryption
Version 3
Download PEM (cert) PEM (chain)
Fingerprints
SHA-256 1C:85:CA:58:CE:95:0B:B7:C2:40:49:89:2B:E1:5C:65:E6:98:F2:9A:88:6A:AF:86:2E:9F:B5:29:77:94:AA:6F
SHA-1 A2:B9:B9:F6:16:08:E9:12:8C:E7:A3:1E:2E:97:AD:D6:7D:AB:CE:43
Basic Constraints
Certificate Authority No
Key Usages
Purposes Digital Signature
Extended Key Usages
Purposes Server Authentication, Client Authentication

Getting a Certificate for your Website

Let's Encrypt is a Certificate Authority that provides free X.509 TLS certificates:

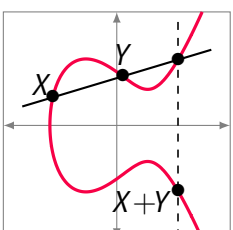
https://letsencrypt.org/getting-started/

Certificates are valid for 90 days and automatically renewed:

https://certbot.eff.org/

ECC = Elliptic Curve Cryptography

- $\mathbb{Z}_p^*$ is not the only useful group for the Discrete Logarithm Problem.
- An attractive alternative is the Elliptic Curve group, where each element is not an integer but a 2-dimensional point with two integer coordinates. The group operation is addition with special point addition formulas.



EC Discrete Logarithm Problem (ECDLP)

Given points P, Q on an elliptic curve with

$$Q = k \cdot P = \underbrace{P + P + \ldots + P}_{k \text{ times}},$$

Find k.

Security Level of Public-Key Schemes

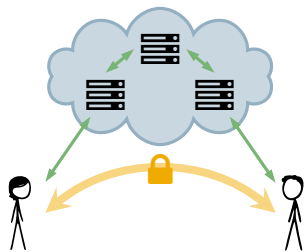
ECDH is the Elliptic-Curve version of Diffie-Hellman key exchange.
(EC)DSA is a signature based on the (Elliptic-Curve) Discrete Logarithm Problem.

Estimated security levels for different modulus bitsizes (NIST SP 800-57):

Security level	RSA	DH, DSA	ECDH, ECDSA
80 bits	1024	1024	160
112 bits	2048	2048	224
128 bits	**3072**	**3072**	**256**
192 bits	7680	7680	384
256 bits	15360	15360	512

Privacy-Enhancing Technologies (PETs) & Protocols

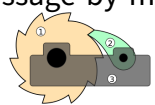
- TLS protects confidentiality and authenticity of communication between the two connection endpoints (client and server).
- Often, we need more complex properties to protect privacy
- Example: End-to-End encryption



End-to-End Encryption (E2EE) Example: Signal

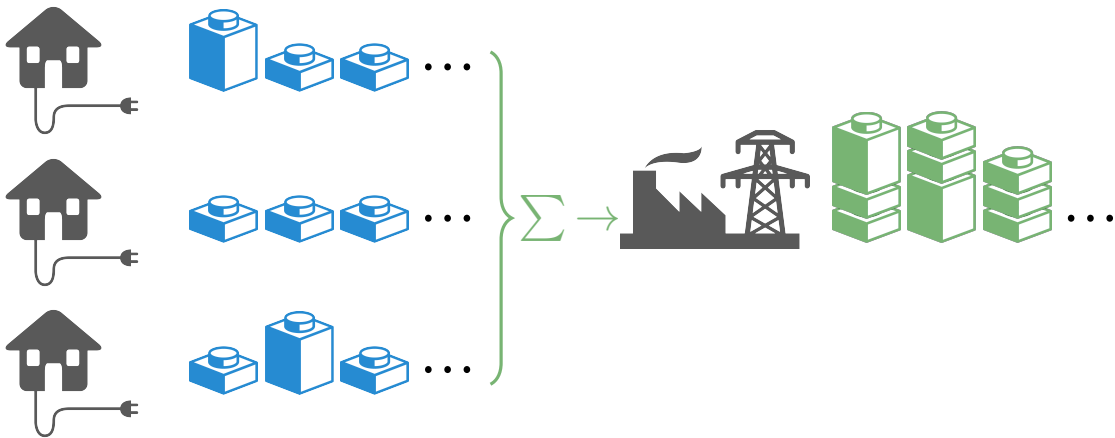
Details here

- E2EE for voice calls and instant messaging + group chats
- Security properties: confidentiality, integrity, authentication, forward secrecy, post-compromise security, participant consistency, destination validation, causality preservation, message unlinkability, message repudiation, participation repudiation, asynchronicity, …
  - How to authenticate and prevent man-in-the-middle attacks?
    - "verify safety number" = hash(public key) (details)
  - How to provide forward & backward secrecy message-by-message?
    - "double ratchet" (update keys with KDF+DH)
    - "triple DH" (mix of DH, DHE)

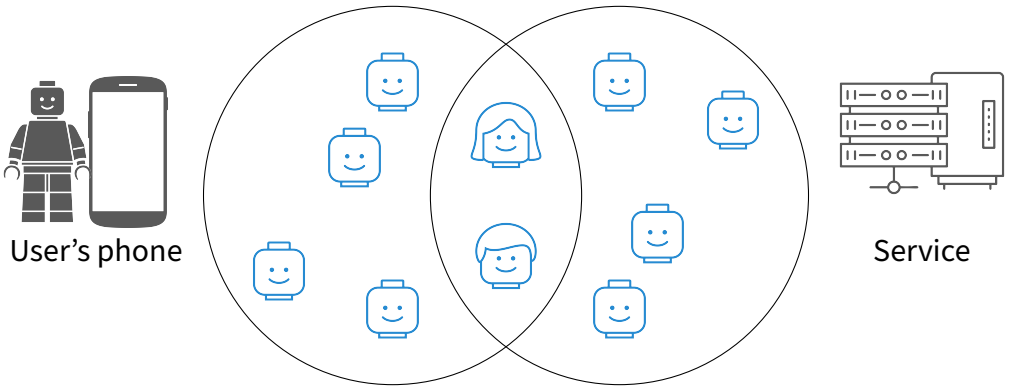


  - How to contact new people asynchronously?
    - "prekey" = a few ephemeral DH public keys stored at server

PET Example 1: Secure Multiparty Computation



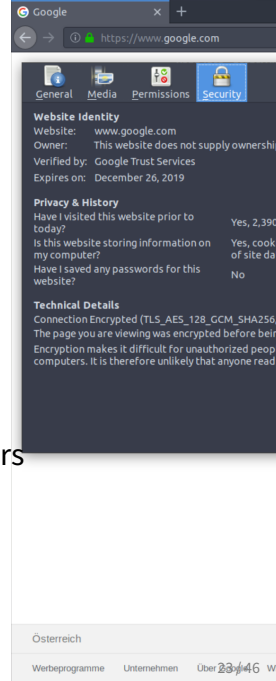
PET Example 2: Private Set Intersection



Crypto Security

Where Do Keys Come From? (1)

- In most cases, they are generated randomly
  - Short-term keys (key exchange, …)
  - Long-term keys (signing, 2FA devices, …)
- High-quality random numbers are essential
  - ✓ Use Cryptographically Secure Random Number Generators (CSPRNGs) like `/dev/urandom`, `OpenSSL` library, …
  - ✗ DON'T use "cheap" random numbers, low-quality seeds, PRNGs without guarantees like `std::random_device()`, non-secure updates like Mersenne Twister, …



Random Number Generators – 2 Types

Nondeterministic hardware source

A hardware random number generator (HRNG) or true random number generator (TRNG) is a device that generates random numbers from a physical process (such as quantum effects or other microscopic effects), rather than by means of an algorithm.

Deterministic pseudorandomness

A pseudorandom number generator (PRNG) or deterministic random bit generator (DRBG) is an algorithm for generating a sequence of numbers whose properties approximate the properties of sequences of random numbers. This sequence is not truly random, but completely determined by an initial value, the seed. A PRNG can be built from cryptographic schemes like stream ciphers or hash functions.

Where Do Keys Come From? (2)

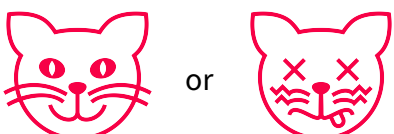
- Sometimes, they are derived from other secrets like passwords or DH secrets
  - Password-Based Key Derivation Functions (PBKDFs), similar to (password) hash functions
  - The quality of the key depends on the quality (entropy) of the password

```
meichlseder@x1tblme ~ % sudo cryptsetup lu
[sudo] password for meichlseder:
LUKS header information
Version:        2
Epoch:          4
Metadata area:  16384 [bytes]
Keyslots area:  16744448 [bytes]
UUID:           087c56a9-a282-42f2-8361-86
Label:          (no label)
Subsystem:      (no subsystem)
Flags:          (no flags)

Data segments:
  0: crypt
        offset: 16777216 [bytes]
        length: (whole device)
        cipher: aes-xts-plain64
        sector: 512 [bytes]

Keyslots:
  0: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2i
        Time cost:  5
        Memory:     1048576
        Threads:    4
        Salt:       81 0d d7 18 01 e4 1d c
                    fc 8c 32 9a 4e 94 a0 a
        AF stripes: 4000
        AF hash:    sha256
        Area offset:32768 [bytes]
        Area length:258048 [bytes]
        Digest ID:  0
  1: luks2
        Key:        512 bits
        Priority:   normal
        Cipher:     aes-xts-plain64
        Cipher key: 512 bits
        PBKDF:      argon2i
        Time cost:  5
        Memory:     1048576
        Threads:    4
        Salt:       f9 df 21 39 44 07 d3 f
                    d1 28 2b 2c 48 5f 8f 0
        AF stripes: 4000
        AF hash:    sha256
        Area offset:290816 [bytes]
        Area length:258048 [bytes]
        Digest ID:  0
Tokens:
```

Quantum Computing

1 Bit

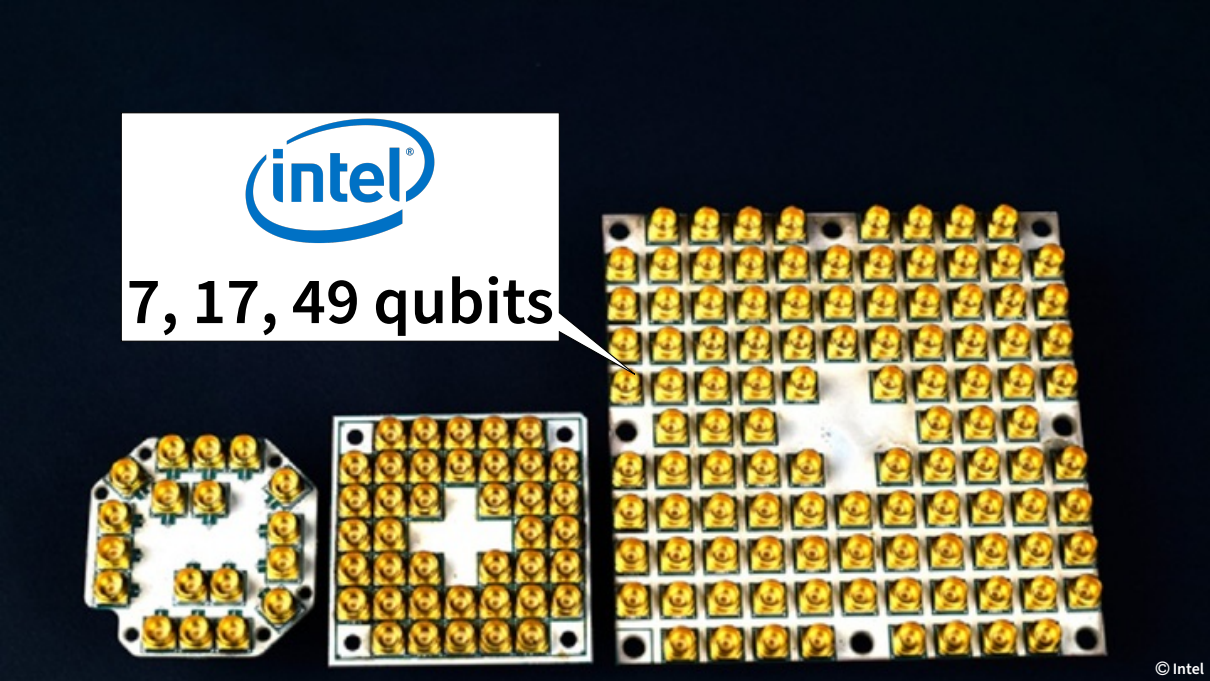
1 of 2 states



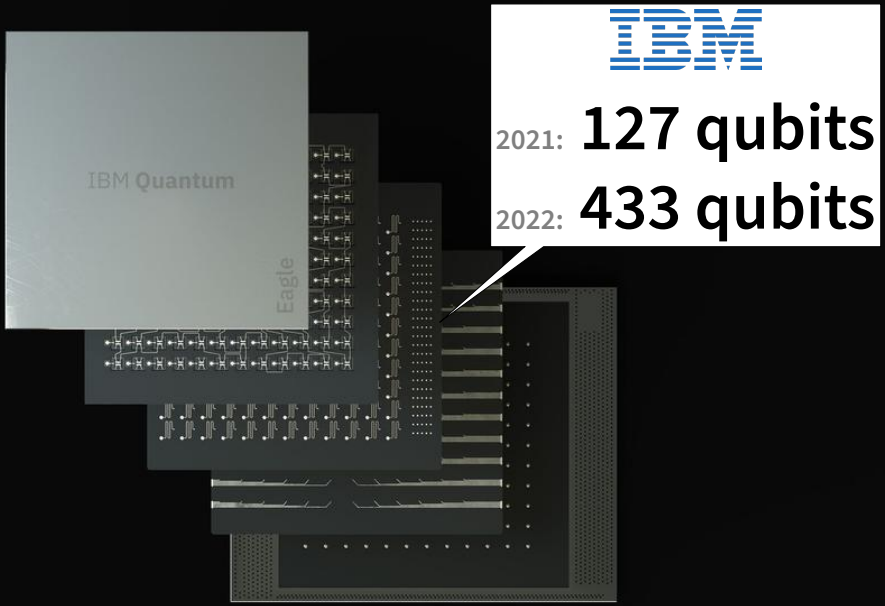
1 Qubit

Superposition of 2 states

- n qubits can be in a superposition of 2^n states
- However, keeping this stable and correcting errors is a huge challenge




Google
72 qubits
Bristlecone
Google
© Google



Quantum Computing & Cryptography

Signatures

Key Exchange

Encryption

Cryptanalysis with Quantum Computers

Fast quantum computers could provide huge speedups in cryptanalysis:

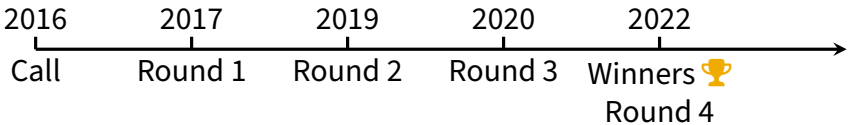
- Shor's Algorithm solves some hard problems in polynomial time:
  - ☠ Integer Factorization Problem (RSA) broken
  - ☠ Discrete Logarithm Problem (ECC, DH) broken

  Don't panic yet: So far, the largest number factored with Shor's algo is 21

- Grover's Algorithm solves some problems in $\mathcal{O}(\sqrt{N})$ instead of $\mathcal{O}(N)$:
  - ⌨ Symmetric crypto: security level is reduced slightly, low impact

The NIST PQ Competition – Standardizing Post-Quantum Crypto

Organized by NIST (US Institute of Standards and Technology)



Goals: Post-quantum secure signature and public-key encryption

NIST received 69 candidates based on different mathematical problems:

Coding theory

Lattices

Hash trees

and more

The NIST PQ Competition – Winners

Key Exchange / Public Key Encryption

- CRYSTALS-Kyber

Signature Schemes

- CRYSTALS-Dilithium
- FALCON
- SPHINCS+ (contributions from)

These are currently being integrated in applications (alone or hybrid with ECC)

Common Security Vulnerabilities (by Language)

Source: Veracode

	.Net	C++	Java	JavaScript	PHP	Python
1	Information Leakage 62.8%	Error Handling 66.5%	CRLF Injection 64.4%	Cross-Site Scripting (XSS) 31.5%	Cross-Site Scripting (XSS) 74.6%	Cryptographic Issues 35.0%
2	Code Quality 53.6%	Buffer Management Errors 46.8%	Code Quality 54.3%	Credentials Management 29.6%	Cryptographic Issues 71.6%	Cross-Site Scripting (XSS) 22.2%
3	Insufficient Input Validation 48.8%	Numeric Errors 45.8%	Information Leakage 51.9%	CRLF Injection 28.4%	Directory Traversal 64.6%	Directory Traversal 20.6%
4	Cryptographic Issues 45.9%	Directory Traversal 41.9%	Cryptographic Issues 43.3%	Insufficient Input Validation 25.7%	Information Leakage 63.3%	CRLF Injection 16.4%
5	Directory Traversal 35.4%	Cryptographic Issues 40.2%	Directory Traversal 30.4%	Information Leakage 22.7%	Untrusted Initialization 61.7%	Insufficient Input Validation 8.3%
6	CRLF Injection 25.3%	Code Quality 36.6%	Credentials Management 26.5%	Cryptographic Issues 20.9%	Code Injection 48.0%	Information Leakage 8.3%
7	Cross-Site Scripting (XSS) 24.0%	Buffer Overflow 35.3%	Cross-Site Scripting (XSS) 25.2%	Authentication Issues 14.9%	Encapsulation 48.0%	Server Configuration 8.1%

What are Common Crypto Failures? I

- Using no crypto (or insufficient crypto)
  - Example: encrypting, but not authenticating
  - Real-world culprit: VMWare View
- Using obsolete crypto
  - Examples: `MD5`, `RC4`, `DES`, 512-bit `RSA`, ...
  - Real-world culprits: Insecure defaults in PHP's `crypt()`, Downgrade attacks like `DROWN`, `FLAME` malware, ...

What are Common Crypto Failures? II

- Building homebrew protocols based on textbook crypto
  - Examples: Textbook `RSA` without padding, `AES` in `ECB` mode, homebrew ciphers
  - Real-world culprits: `Keeloq` and other car key systems

- Using backdoored crypto
  - Examples: `Dual_EC_DRBG`
  - Real-world culprits: RSA's BSafe, Juniper Networks' ScreenOS firmware

What are Common Crypto Failures? III

- Improper key management
  - Example: Same master password on all devices; hardcoded keys; certificates not checked
  - Real-world culprit: Philips Hue smart lightbulbs; GitHub key search

- Improper password storage
  - Example: Storing passwords in plain or hashed without a salt, with `MD5`/`SHA-1`
  - Real-world culprit: LinkedIn hack 2012

What are Common Crypto Failures? IV

- Nonce misuse
  - Examples: Reusing constant values for nonces, IVs, or ephemeral keys
  - Real-world culprits: Sony PS3, Western Digital self-encrypting drives

- Bad randomness, insufficient entropy
  - Examples: `rand()`, `time()`, 32-bit seeds, passwords, …
  - Real-world culprit: Western Digital self-encrypting drives, WEP

Conclusion

Conclusion

- Current applications build on well-understood crypto
  - Secure, up-to-date standards
  - Efficient and readily available
- Modern crypto research opens a multitude of new opportunities
  - Privacy-preserving technologies
  - Very-long-term (quantum) security
  - …

Some example exam questions from previous years I

[2 P] (b) Name **two** of the main **security properties** of assets considered in information security.

- _____ • _____

[2 P] (c) For one of these two security properties, **give an example** of one cryptographic scheme that **protects** this property and one that **does not**.

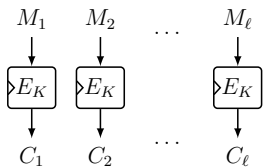
Some example exam questions from previous years II

[3 P] (b) Complete the following specification of the **Diffie–Hellman** protocol:

- Public system parameters: A large prime p and a generator α of $\mathbb{Z}_p^*$
- Alice chooses and computes: _____
- Alice sends to Bob: _____
- Bob chooses and computes: _____
- Bob sends to Alice: _____
- Alice computes: $K =$ _____
- Bob computes: $K =$ _____

Some example exam questions from previous years III

[2 P] (c) Consider the **mode of operation** illustrated below to use AES (denoted E_K) for encryption. Why is this mode **not secure**?



Some example exam questions from previous years IV

[4 P] (c) *Check **all** correct answers. There may be several correct answers per question. +1 point for correct choice, −1 point for wrong choice, 0 points for no choice.*

Which of the following schemes provide **Message Authentication**?

- ☐ Encryption
- ☐ Signature
- ☐ MAC
- ☐ Authenticated Encryption (AEAD)

Which of the following schemes and protocols provide **Entity Authentication**?

- ☐ Hash function
- ☐ Password protocol
- ☐ Encryption
- ☐ Challenge-response protocol

Which of the following schemes allow **only the owner of the secret key to verify** authenticity?

- ☐ MAC
- ☐ Authenticated Encryption (AEAD)
- ☐ Hash function
- ☐ Signature