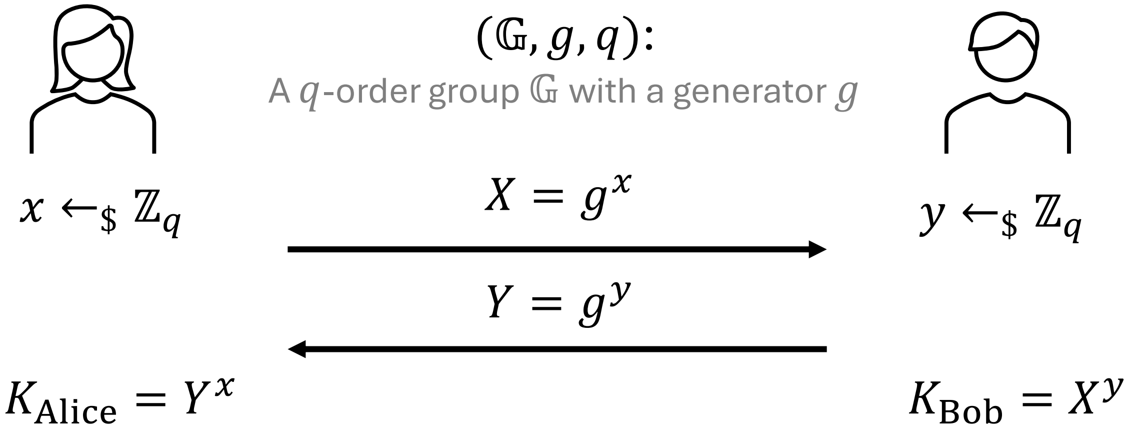
Cryptography Engineering

- Lecture 10 (Jan 22, 2024)
- Today's notes:
  - Some attacks on Cryptosystems (and how to prevent them)
  - Towards Post-Quantum Cryptography

Attacks using Invalid Inputs

- The adversary sends data that violates the protocol or data format.
- **Example: DHKE**



$(\mathbb{G}, g, q)$:

A q-order group $\mathbb{G}$ with a generator g

$x \leftarrow_\$ \mathbb{Z}_q$

$X = g^x$

$y \leftarrow_\$ \mathbb{Z}_q$

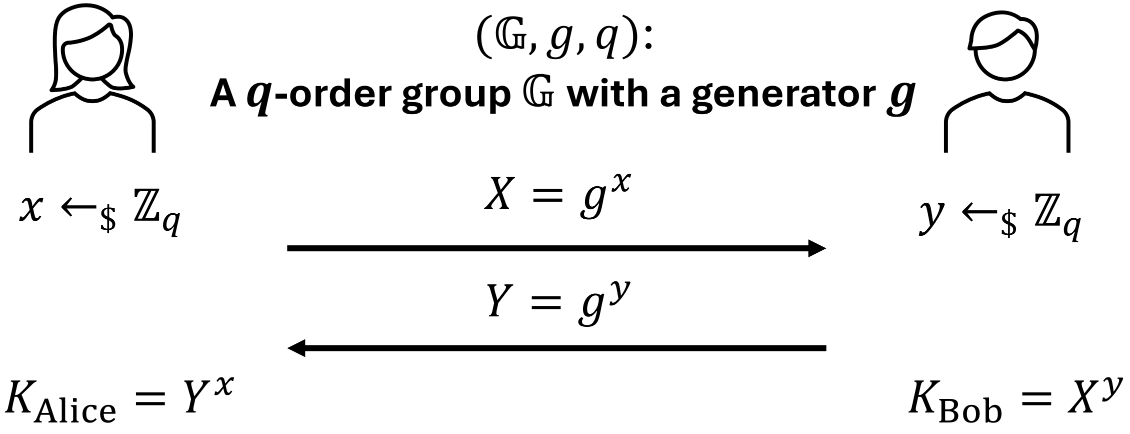
$Y = g^y$

$K_{\text{Alice}} = Y^x$

$K_{\text{Bob}} = X^y$

Attacks using Invalid Inputs

- The adversary sends data that violates the protocol or data format.
- **Example: DHKE**



$(\mathbb{G}, g, q)$:

A q-order group $\mathbb{G}$ with a generator g

$x \leftarrow_\$ \mathbb{Z}_q$

$y \leftarrow_\$ \mathbb{Z}_q$

$X = g^x$

$Y = g^y$

$K_{\text{Alice}} = Y^x$

$K_{\text{Bob}} = X^y$

- The security holds **if the protocol runs on specific groups**
- **What if we use an element outside the group $\mathbb{G}$?**

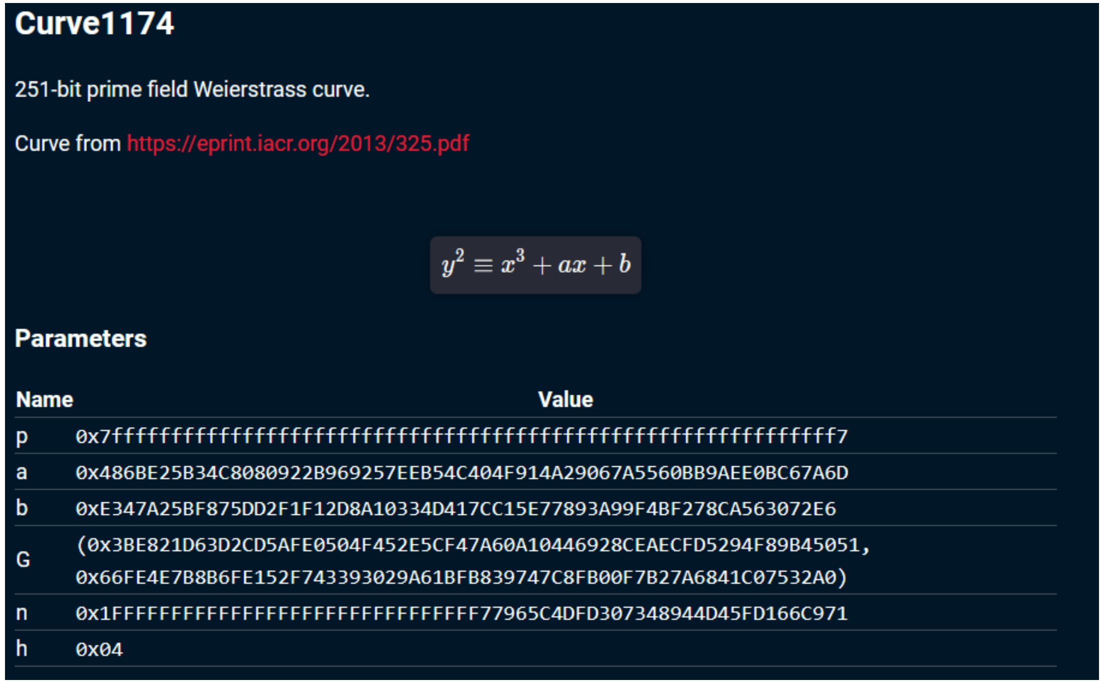
Attacks using Invalid Inputs

- The adversary sends data that violates the protocol or data format.
- **Example: DHKE**

$(\mathbb{G}, g, q)$:

A $\boldsymbol{q}$-order group $\mathbb{G}$ with a generator $\boldsymbol{g}$

- $\mathbb{G}$ can be a subgroup of another group $\mathbb{G}'$
- Co-factor: $|\mathbb{G}'|/|\mathbb{G}|$ (the h value on the RHS figure)



Curve1174

251-bit prime field Weierstrass curve.

Curve from https://eprint.iacr.org/2013/325.pdf

$$y^2 \equiv x^3 + ax + b$$

Parameters

Name	Value
p	0x7ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff7
a	0x486BE25B34C8080922B969257EEB54C404F914A29067A5560BB9AEE0BC67A6D
b	0xE347A25BF875DD2F1F12D8A10334D417CC15E77893A99F4BF278CA563072E6
G	(0x3BE821D63D2CD5AFE0504F452E5CF47A60A10446928CEAECFD5294F89B45051, 0x66FE4E7B8B6FE152F743393029A61BFB839747C8FB00F7B27A6841C07532A0)
n	0x1FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF77965C4DFD307348944D45FD166C971
h	0x04

Source: https://neuromancer.sk/std/other/Curve1174

Attacks using Invalid Inputs

- The adversary sends data that violates the protocol or data format.
- **Example: DHKE**

$(\mathbb{G}, g, q)$:

A q-order group $\mathbb{G}$ with a generator g

- $\mathbb{G}$ can be a subgroup of another group $\mathbb{G}'$
- Co-factor: $|\mathbb{G}'|/|\mathbb{G}|$ (the h value on the RHS figure)
- **Use the co-factor to check group membership**

$$\xrightarrow{X\, =\, g^x}$$

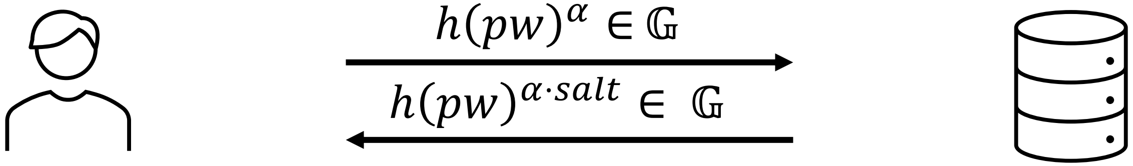
Check $X^h = 1$?
// 1 is the identity group element
If so, reject
else:
$y \leftarrow_\$ \mathbb{Z}_q$

Attacks using Invalid Inputs

- **Toy Example of attacking OPAQUE:**

$(\mathbb{G} \subset \mathbb{G}', g, q, h = 2)$:

A $\boldsymbol{q}$-order group $\mathbb{G}$ with a generator $\boldsymbol{g}$, and $|\mathbb{G}'|/|\mathbb{G}| = h$



$h(pw)^{\alpha} \in \mathbb{G}$

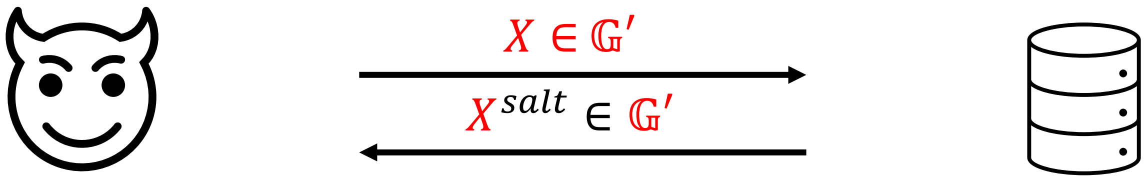
$h(pw)^{\alpha \cdot salt} \in \mathbb{G}$

Attacks using Invalid Inputs

- **Toy Example of attacking OPAQUE:**

$(\mathbb{G} \subset \mathbb{G}', g, q, h = 2)$:

A $\boldsymbol{q}$-order group $\mathbb{G}$ with a generator $\boldsymbol{g}$, and $|\mathbb{G}'|/|\mathbb{G}| = h$

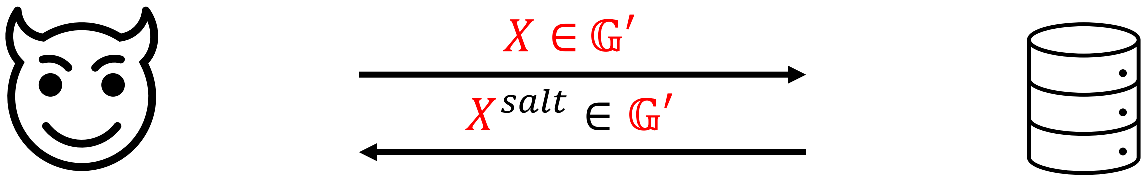


Find an element X s.t. X's order is 2

Attacks using Invalid Inputs

- **Toy Example of attacking OPAQUE:**

$(\mathbb{G} \subset \mathbb{G}', g, q, h = 2)$:
A $\boldsymbol{q}$-order group $\mathbb{G}$ with a generator $\boldsymbol{g}$,
and $|\mathbb{G}'|/|\mathbb{G}| = h$



Find an element X s.t.
X's order is 2

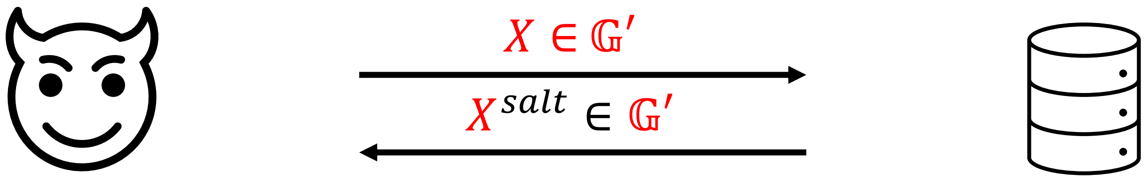
Little Algebra:
If X's order is 2, then $X^r = X^{(r \bmod 2)}$ **=> We can determine the parity of the salt:** $salt$ is an odd/even number if $X^{salt} = X$

Attacks using Invalid Inputs

- **Toy Example of attacking OPAQUE:**

$(\mathbb{G} \subset \mathbb{G}', g, q, h = 2)$:

A $\boldsymbol{q}$-order group $\mathbb{G}$ with a generator $\boldsymbol{g}$,
and $|\mathbb{G}'|/|\mathbb{G}| = h$



Find an element X s.t.
X's order is 2

Exercise: Extend it to more general cases

Attacks using Invalid Inputs

- Other Example:
    - Invalid Curve Attacks (e.g. ECDSA): Using insecure curves.
    - Invalid public keys
    - …

- Lessons: Follow the standards(/specifications/…), and keep updating with them…

Downgrade Attacks

- Exploit vulnerabilities in compatibility or protocol negotiation to downgrade cryptographic protocols to weaker or obsolete versions.

- Example: TLS cipher cuite negotiation
  - TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256 (secure)
  - TLS_RSA_WITH_RC4_128_SHA (no forward secrecy)

- Lessons: Use the latest protocol version (such as TLS 1.3), disable insecure or outdated protocols/suites on both sides.

More Examples about Reuse

- Previous Example: Randomness Reuse in the DSA signature => Recovery of secret key

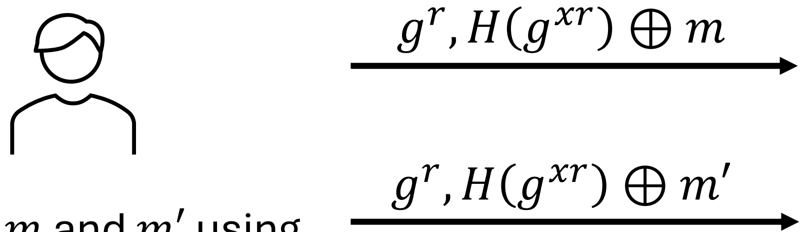
- Why should we **not** reuse randomness?

- ➢ An informal principle: Security of cryptosystem comes from *the secret key* and *the randomness*
  - ➢ Secret key: **High entropic, the "source" of security, …**
  - ➢ Randomness/nonce/salt: **Independency when using the same key, Freshness, …**

More Examples about Reuse

- Example: Reuse randomness in the Hashed ElGamal Encryption

```
ElGamalEnc(public_key = $g^x$, plaintext = $m$)
// $(\mathbb{G}, g, q)$: A $q$-order group $\mathbb{G}$ with a generator $g$
1. $r \leftarrow_{\$} \mathbb{Z}_q$
2. $c_0 = g^r$
3. $c_1 = H(g^{xr}) \oplus m$
4. Return $(c_0, c_1)$
```



$g^r, H(g^{xr}) \oplus m$

$g^r, H(g^{xr}) \oplus m'$

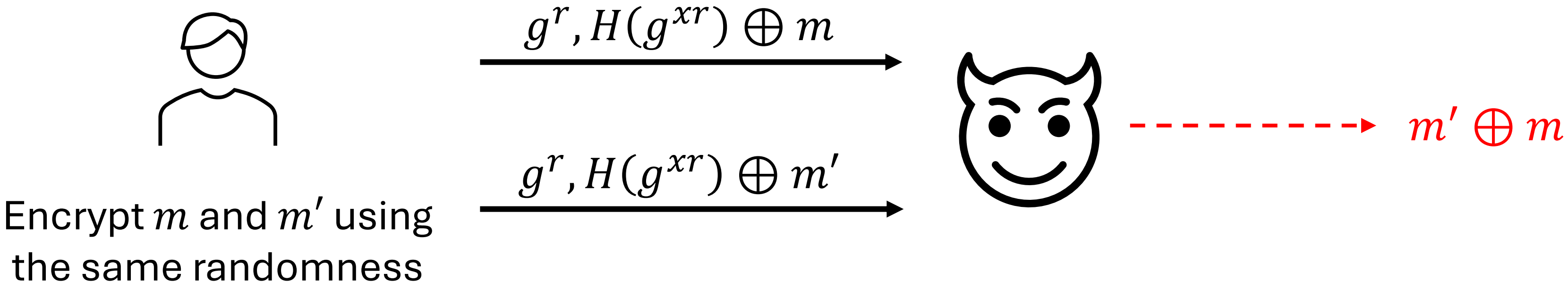
Encrypt m and m' using the same randomness

More Examples about Reuse

- Example: Reuse randomness in the Hashed ElGamal Encryption

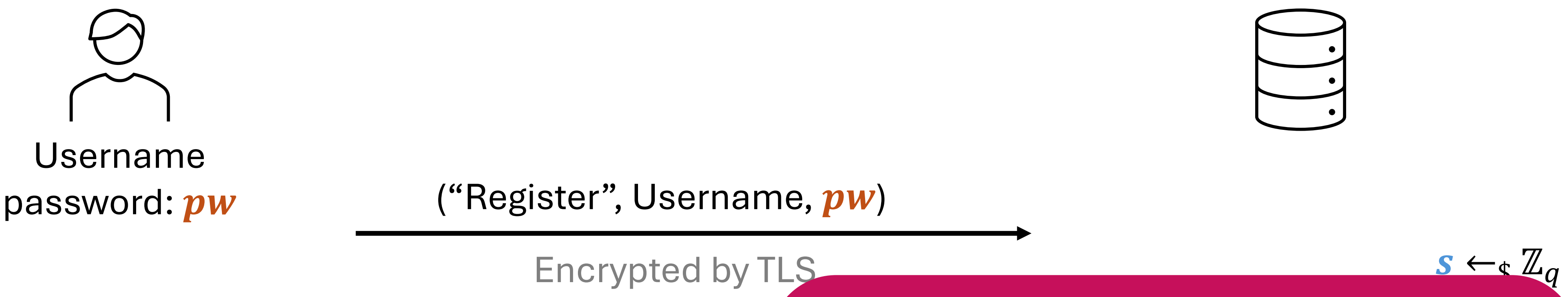
```
ElGamalEnc(public_key = $g^x$, plaintext = $m$)
// ($\mathbb{G}, g, q$): A $q$-order group $\mathbb{G}$ with a generator $g$

1. $r \leftarrow_\$ \mathbb{Z}_q$
2. $c_0 = g^r$
3. $c_1 = H(g^{xr}) \oplus m$
4. Return $(c_0, c_1)$
```



Encrypt m and m' using the same randomness

More Examples about Reuse



Username
password: $\boldsymbol{pw}$

("Register", Username, $\boldsymbol{pw}$)

Encrypted by TLS

$\boldsymbol{s} \leftarrow_{\$} \mathbb{Z}_q$

$(lpk_c,$

Then the server store {
user: **Username** // … as index
salt: r
~~*password_file:* $\boldsymbol{rw}$~~
server_k_bundle: $\boldsymbol{lpk_c, lpk_s, lsk_s}$
client_enc_k_bundle: **enc_client_keys**
… // Auxiliary information
} in the password database

Correction: The server does not store rw

More Examples about Reuse

- Examples: Reuse salt in OPAQUE
- Suppose that Alice's password is pw_A, Bob's password is pw_B, and the password files stored in the server are:

Username:	Bob	Username:	Alice
salt:	r	salt:	r
enc_AKE_keys:	$\mathrm{AEAD}_{rw_B}(\ldots)$	enc_AKE_keys:	$\mathrm{AEAD}_{rw_A}(\ldots)$

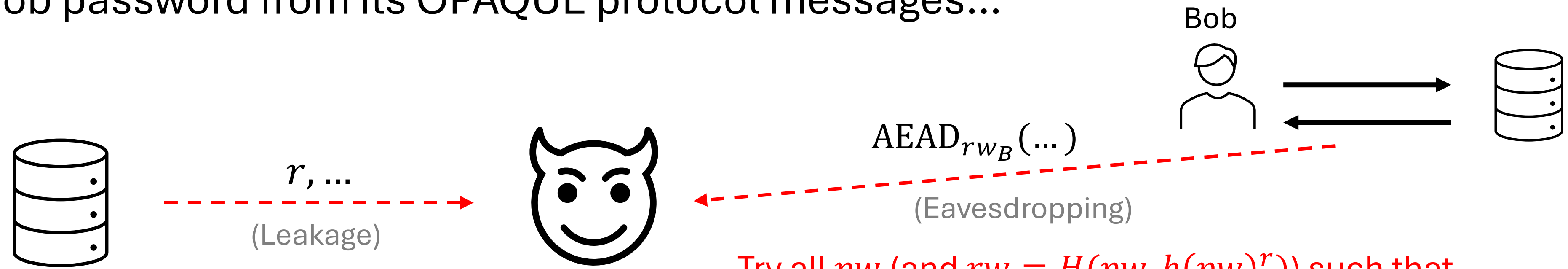
- Is it secure? Why?

More Examples about Reuse

- Examples: Reuse salt in OPAQUE
- Suppose that Alice's password is pw_A, Bob's password is pw_B, and the password files stored in the server are:

Username:	Bob	Username:	Alice
salt:	r	salt:	r
enc_AKE_keys:	$\mathrm{AEAD}_{rw_B}(\dots)$	enc_AKE_keys:	$\mathrm{AEAD}_{rw_A}(\dots)$

- **Potential risks:** If Alice's password file is leaked, then the adversary can launch offline attacks to recover Bob password from its OPAQUE protocol messages…



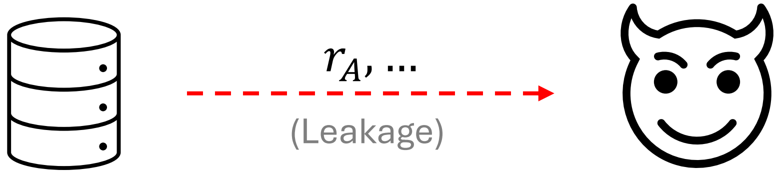
Try all pw (and $rw = H(pw, h(pw)^r)$) such that AEAD.Dec does not output rejection…

More Examples about Reuse

- Examples: Single-seed-derived salt in OPAQUE
- Suppose that the server has a random $seed$, Alice's password is pw_A, Bob's password is pw_B, and the password files stored in the server are:

Username:	Bob	Username:	Alice
salt:	$r_B = \mathrm{PRF}(seed,\ \text{“Bob”})$	salt:	$r_A = \mathrm{PRF}(seed,\ \text{“Alice”})$
enc_AKE_keys:	$\mathrm{AEAD}_{rw_B}(\ldots)$	enc_AKE_keys:	$\mathrm{AEAD}_{rw_A}(\ldots)$

- Suppose that the seed is stored separately in some secure way...
- **Is it secure?**



More Examples about Reuse

- Other examples:
  - Reuse randomness in Schnorr/Schnorr-like signature schemes…
  - Reuse of IV in the AES-GCM mode, or short IV…
  - Reuse randomness in SRP
  - …

Side-Channel Attacks

- Side-channel information: By-product information when the system runs cryptographic algorithms.
    - E.g., time, power consumption, cache access patterns, ...

- Example:
    - Timing Attacks
    - Cache Attacks
    - ...

- An Example of Timing Attack: A website checks a user’s password character by character, returning an error as soon as it finds the first mismatch.

- Lessons: Use constant-time algorithms, masking sensitive operations, ...

Towards Post-Quantum Cryptography

- All previous attack examples are about **wrong implementations** of cryptographic algorithms, but not about the algorithms themselves…
    - ➢ Example: Breaking the ElGamal encryption => Solving DH problems…

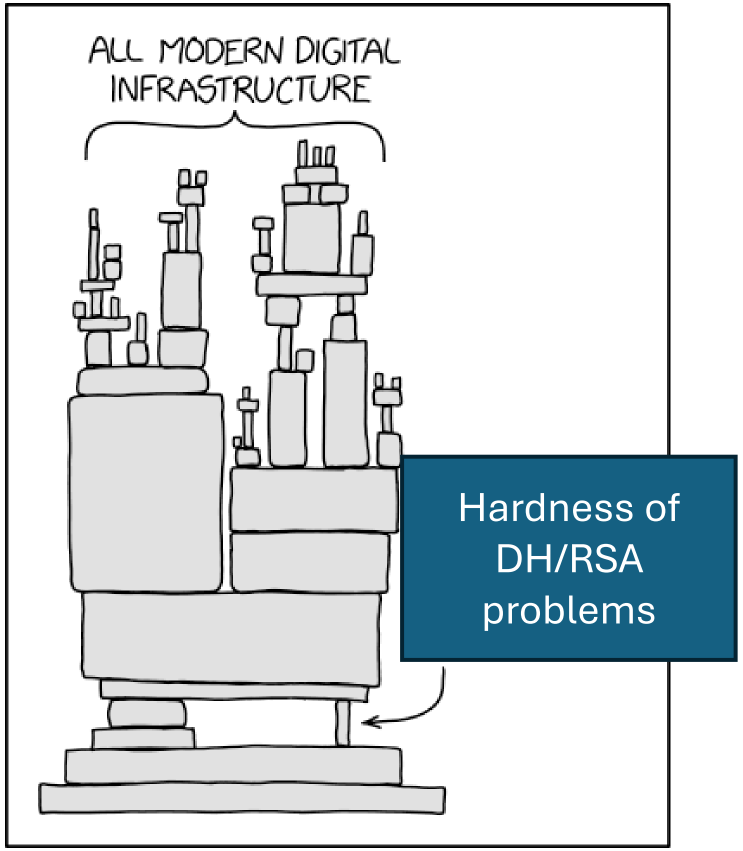
Towards Post-Quantum Cryptography

- All previous attack examples are about wrong implementations of cryptographic algorithms, but not about the algorithms themselves…
    - ➢ Example: Breaking the ElGamal encryption => Solving DH problems…

- Modern cryptography builds on **hardness assumptions**:
    - ElGamal encryption, DHKE, DSA, TLS 1.3, and others all rely on the hardness of Diffie-Hellman or RSA problems…
    - We assume these problems are hard to solve (i.e., there is no polynomial-time algorithm).

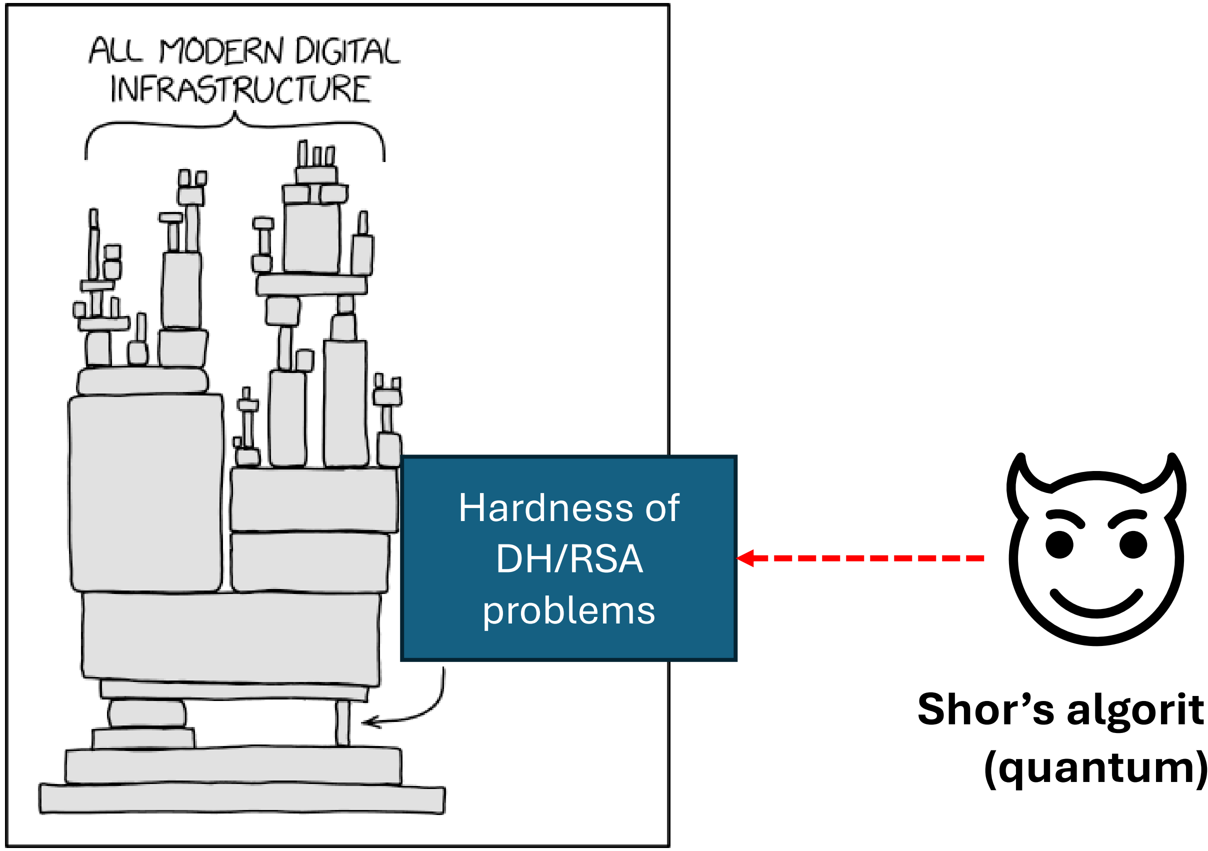
- What if these assumptions are broken?

Towards Post-Quantum Cryptography


ALL MODERN DIGITAL INFRASTRUCTURE
Hardness of DH/RSA problems

Source: xkcd/2347 and Nadia
Heninger's talk in PKC2024

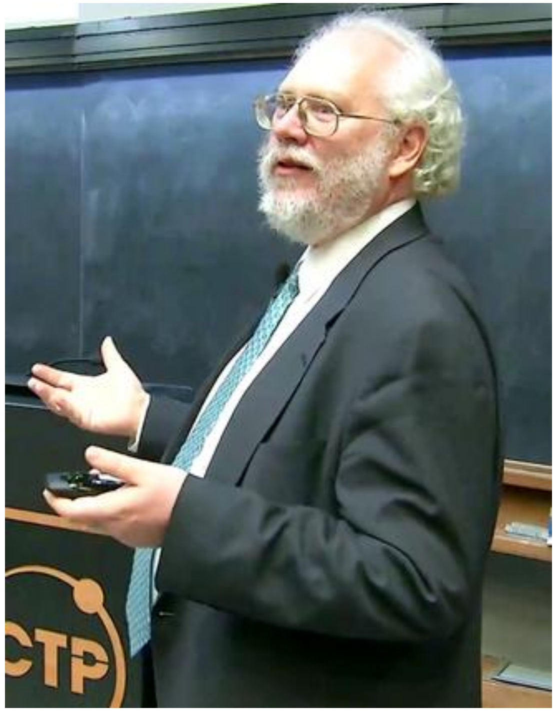
Towards Post-Quantum Cryptography



Hardness of DH/RSA problems

Shor's algorithm (quantum)

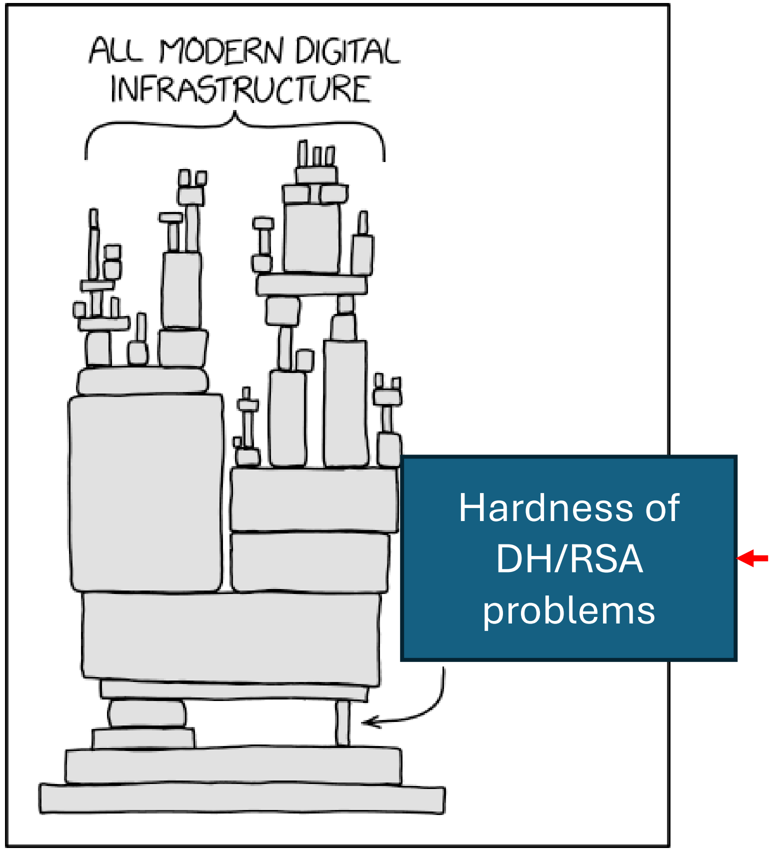
Source: xkcd/2347 and Nadia Heninger's talk in PKC2024



Peter Williston Shor

(image from Wikipedia)

Towards Post-Quantum Cryptography



Hardness of DH/RSA problems

Source: xkcd/2347 and Nadia Heninger's talk in PKC2024

Recent progress in Quantum Computers/Mechanisms...

Shor's algorithm

Towards Post-Quantum Cryptography

- New Direction: Post-Quantum Cryptography
  - Cryptographic algorithms run on classical computers, but **remain secure against future quantum computers**...
- Still follow the methodology of modern cryptography: Assumptions => Schemes.

Towards Post-Quantum Cryptography

- New Direction: Post-Quantum Cryptography
  - Cryptographic algorithms run on classical computers, but **remain secure against future quantum computers...**
- Still follow the methodology of modern cryptography: Assumptions => Schemes.

- **Hardness Assumptions even against quantum adversaries:**
  - **Lattices**
  - Isogeny (of Elliptic Curves)
  - Code-based
  - ...

- Standardization in progress (https://csrc.nist.gov/Projects/post-quantum-cryptography/news)

Towards Post-Quantum Cryptography

- New Direction: Post-Quantum Cryptography
  - Cryptographic algorithms run on classical computers, but **remain secure against future quantum computers**...
- Still follow the methodology of modern cryptography: Assumptions => Schemes.

- **Hardness Assumptions even against quantum adversaries:**
  - **Lattices**
  - Isogeny (of Elliptic Curves)
  - Code-based
  - ...

The last three lectures:
Post-Quantum Cryptography
with a focus on **Lattice-based Cryptography**

- Standardization in progress (https://csrc.nist.gov/Projects/post-quantum-cryptography/news)

Homework

- (1 point) Extend the toy example of attacking OPAQUE using small-order element to the case that h = 4. What information will be revealed in this case?
- (1 point) Extend the toy example of attacking OPAQUE using small-order element to the case that h = 2^{λ} where $\lambda \approx 16{\sim}32$.

- (2 point) Try implementing pre-computation attacks (the complexity should be O(log |D|)).
  - Suppose that the client's password is pw^*, the salt stored in the server is $salt^*$, and the password file stored in the database is

$$(salt^*, v = g^{H(salt^*, [user\_name], pw^*)}) \text{ // } salt^*, v \text{ is in the example code}$$

  - Suppose that you get the salt and know the password is in a dictionary D (in the example code).