DATA PRIVACY AND SECURITY

Prof. Daniele Venturi

Master's Degree in Data Science
Sapienza University of Rome

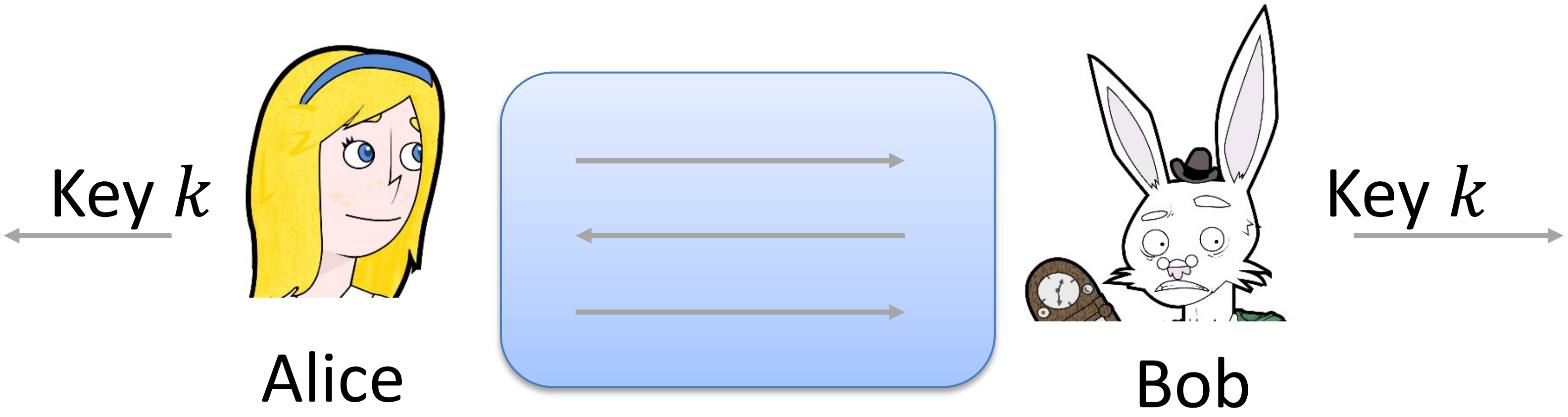
CIS SAPIENZA

RESEARCH CENTER FOR CYBER INTELLIGENCE
AND INFORMATION SECURITY

CHAPTER 3:

Key Exchange

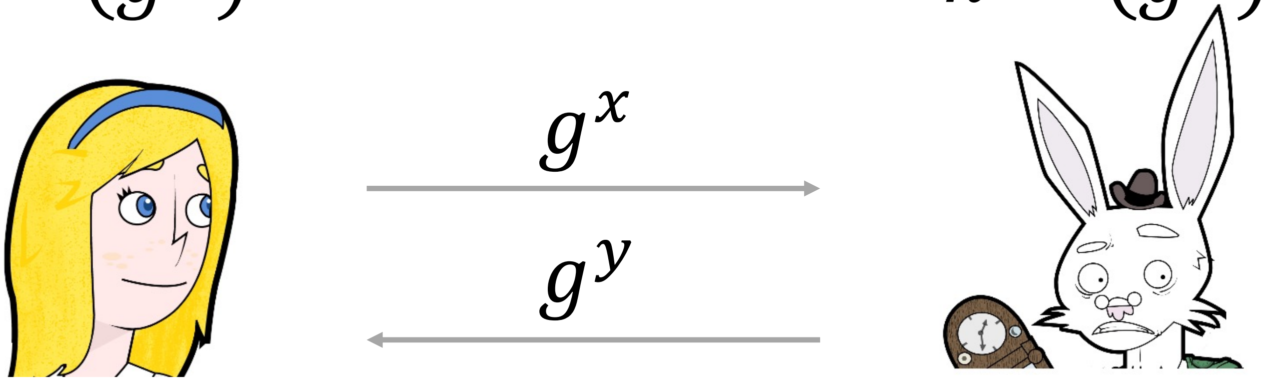
Key Exchange Protocols



Key Exchange Protocol

- Allows to **agree** on a key over a **public** channel
  - KE bootstraps **secure communication**
  - KE constitues the **link** between **symmetric** and **asymmetric** cryptography

Diffie-Hellman Key Exchange



$k = (g^y)^x$

$k = (g^x)^y$

g^x

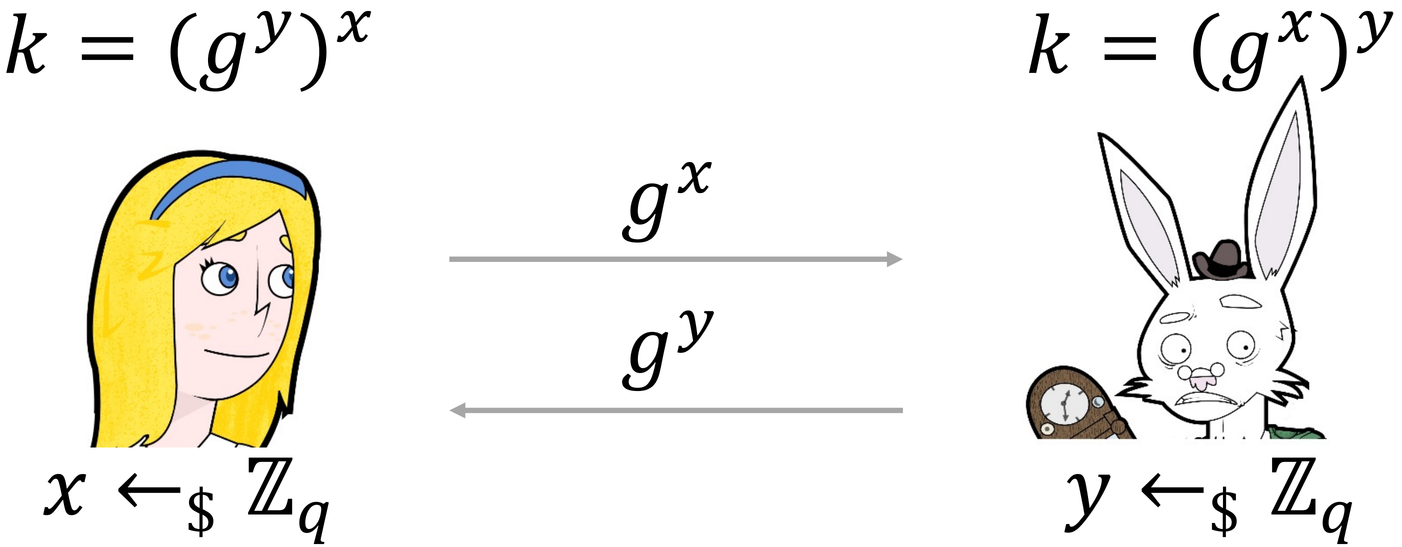
g^y

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

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

- $\mathbb{G}$ is a cyclic group of prime order q, with **generator** g
  - **Passive security** follows from DDH
  - E.g., $\mathbb{G}$ is a subgroup of $\mathbb{Z}_p^*$ where $q|p-1$

Perfect Forward Secrecy



$k = (g^y)^x$

$k = (g^x)^y$

g^x

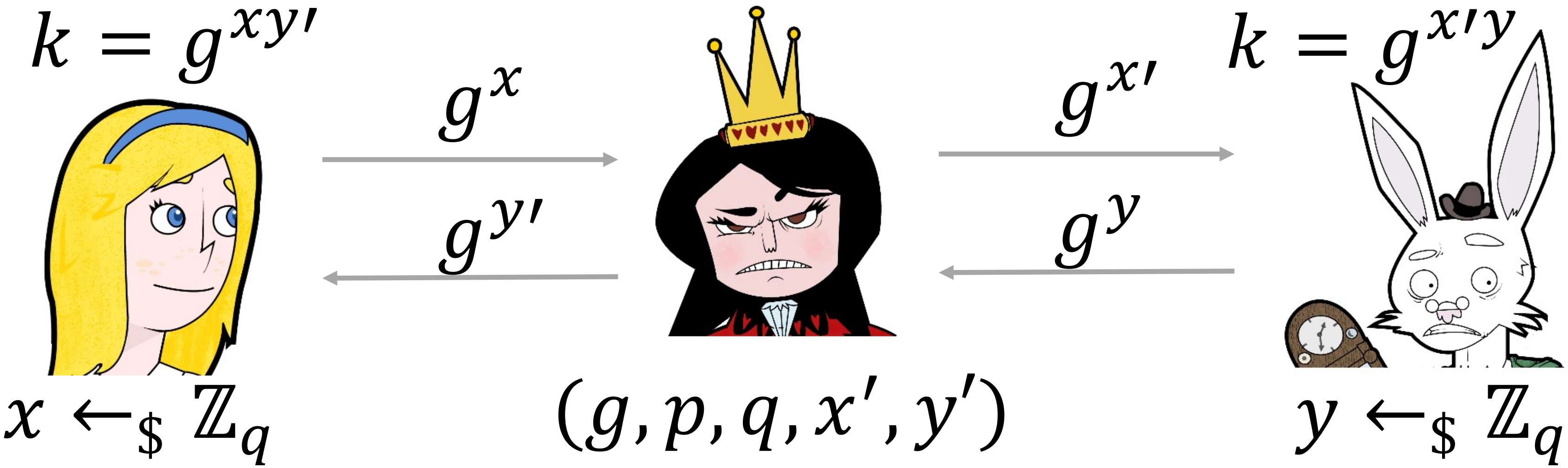
g^y

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

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

- Once the session keys are **destroyed** there is **no way** to recover them
  - Not even the owners (not even at gun point)

(Wo)Man-in-the-Middle Attack



$k = g^{xy'}$

g^x

$g^{y'}$

$g^{x'}$

g^y

$k = g^{x'y}$

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

(g, p, q, x', y')

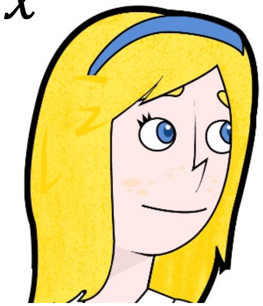
$y \leftarrow_\$ \mathbb{Z}_q$

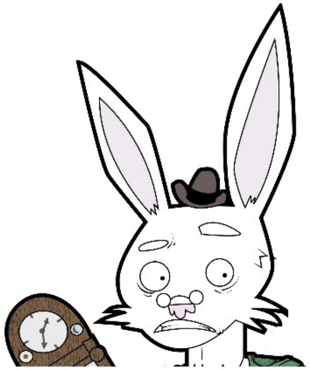
- Eve shares one secret key with **each party**
  - She can decrypt all subsequent communication
- Solution: **Authenticate** messages!
  - Master keys and session keys

Authenticated Key Exchange (AKE)

- Allow two parties to establish a **common secret** in an **authenticated** way
  - Parties should possess **previously established** authentication keys (master keys)
- **Secrecy:** The session key should be indistinguishable from a **random string**
- Additional properties:
  - **Mutual** authentication
  - **Consistency** (honest parties have a **consistent** view of **who** the peers to the session are)

First Attempt





$X = g^x$

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

sk_A, pk_B

$A, X, \mathbf{S}(sk_A, X)$ ⟶

⟵ $B, Y, \mathbf{S}(sk_B, Y)$

$Y = g^y$

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

sk_B, pk_A

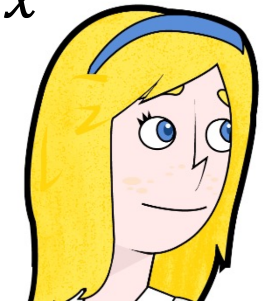
$K = Y^x$

$K = X^y$

- What if Eve ever finds an $(x, g^x, \mathbf{S}(sk_A, X))$?
  - Ephemeral leakage should not allow **long-term impersonation**!

Second Attempt

$X = g^x$



$x \leftarrow_{\$} \mathbb{Z}_q$
sk_A, pk_B

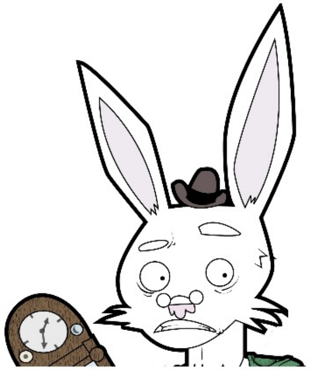
$K = Y^x$

$\xrightarrow{A, X}$

$\xleftarrow{B, Y, \mathbf{S}(sk_B, X||Y)}$

$\xrightarrow{\mathbf{S}(sk_A, X||Y)}$

$Y = g^y$

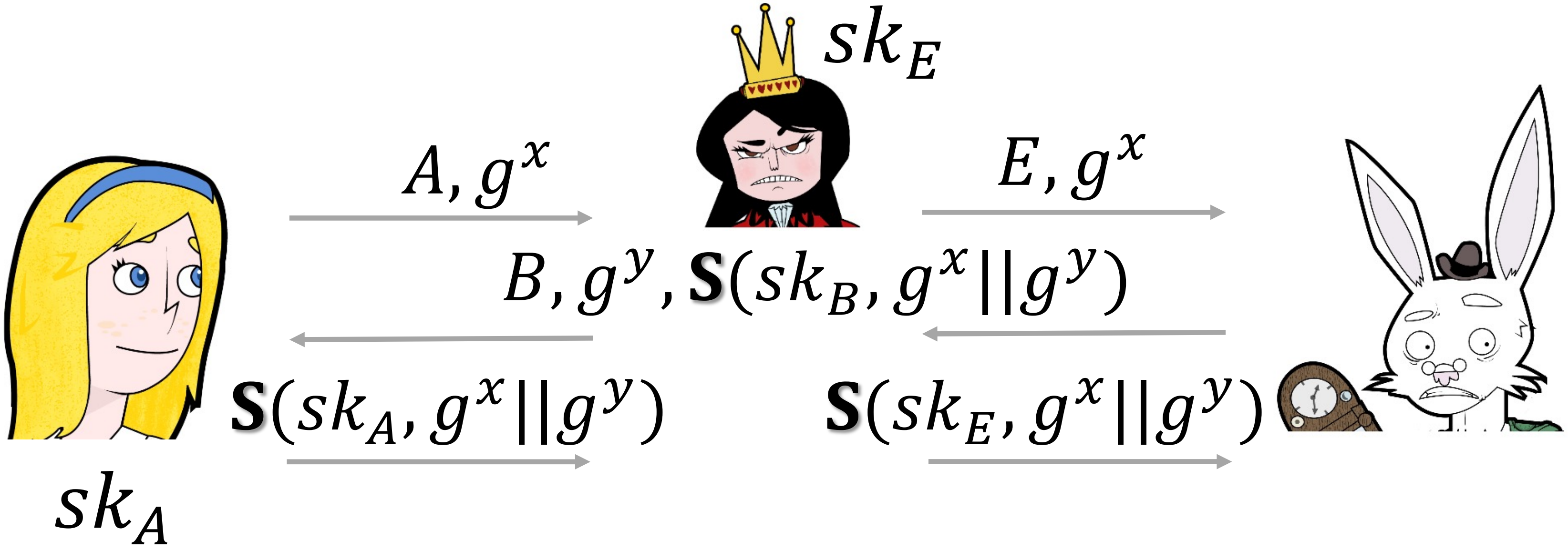


$y \leftarrow_{\$} \mathbb{Z}_q$
sk_B, pk_A

$K = X^y$

- View of the parties at the end of the protocol
  - A: Shared $K = g^{xy}$ with B
  - B: Shared $K = g^{xy}$ with A
  - **Looks fine, but…**

Identity-Misbinding Attack



sk_E

A, g^x

E, g^x

$B, g^y, \mathbf{S}(sk_B, g^x || g^y)$

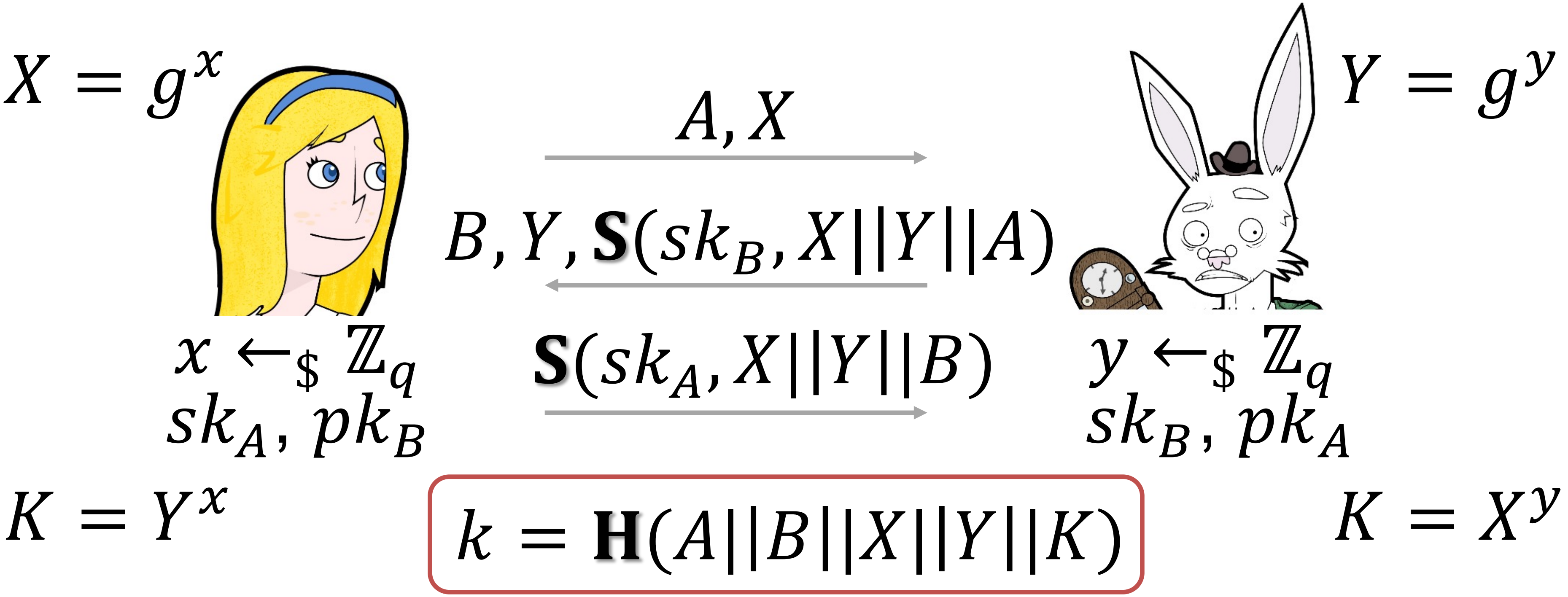
$\mathbf{S}(sk_A, g^x || g^y)$

$\mathbf{S}(sk_E, g^x || g^y)$

sk_A

- Wrong **identity binding**!
  - $A: K \Leftrightarrow B$, but $B: K \Leftrightarrow E$
- Eve doesn't know K, but Bob considers anything coming from Alice **as from Eve**

The ISO 9796 Defense



$X = g^x$

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

sk_A, pk_B

$\xrightarrow{A, X}$

$\xleftarrow{B, Y, \mathbf{S}(sk_B, X||Y||A)}$

$\xrightarrow{\mathbf{S}(sk_A, X||Y||B)}$

$Y = g^y$

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

sk_B, pk_A

$K = Y^x$

$$\boxed{k = \mathbf{H}(A||B||X||Y||K)}$$

$K = X^y$

- Include the **peer identity** under the signature
  - Note that Eve cannot forge $\mathbf{S}(sk_B, X||Y||A)$
  - Avoids previous attack, and can be **proven secure**

Security Desiderata

- Intuitive (e.g., attacker capabilities, secrecy, …)
- Reject **bad** protocols
- Accept **good** protocols
- Ensure security of **applications**
  - **Secure communication** in primis
  - **Composition** and usability
- We will overview the Canetti-Krawczyk (CK) model which is used to analyze many **real-world** KE protocols

Elements of the Definition

- A **two-party** protocol in a **multi-party** setting
- Multiple protocol executions run **concurrently**
  - Each run of a protocol at a party is called **session**
- Sessions are given **unique** names
  - (A, s_A) and (B, s_B) where B is the **intended peer**
  - The **session id** is (A, s_A, B, s_B)
  - Sessions with **corresponding** names like (A, s_A, B, s_B) and (B, s_B, A, s_A) are **matching**
  - At the end, a session outputs the **session id** and the **session key**

The Attacker

- We only assume **unauthenticated** channels
- The adversary
  - Monitors/controls/**modifies** traffic
  - Schedules sessions at will (**interleaving**)
  - May corrupt parties learning **long-term** secrets along with any state information and session keys
  - May issue learning queries for **short-term** information (e.g., session keys or state)
- A session is **exposed** if the owner is corrupted or the adversary issued learning query

The Security Definition

- Completed **matching** sessions output the same key (**correctness**)
- The attacker learns **nothing** about **unexposed** sessions
  - **Test session** chosen by the adversary
  - Attacker is given either the **honest** key or a **randomly generated** key and can’t distinguish
  - Key **confirmation** can be added to the definition
- Note: Never use **session keys** as part of the KE protocol itself (e.g., TLS 1.2)

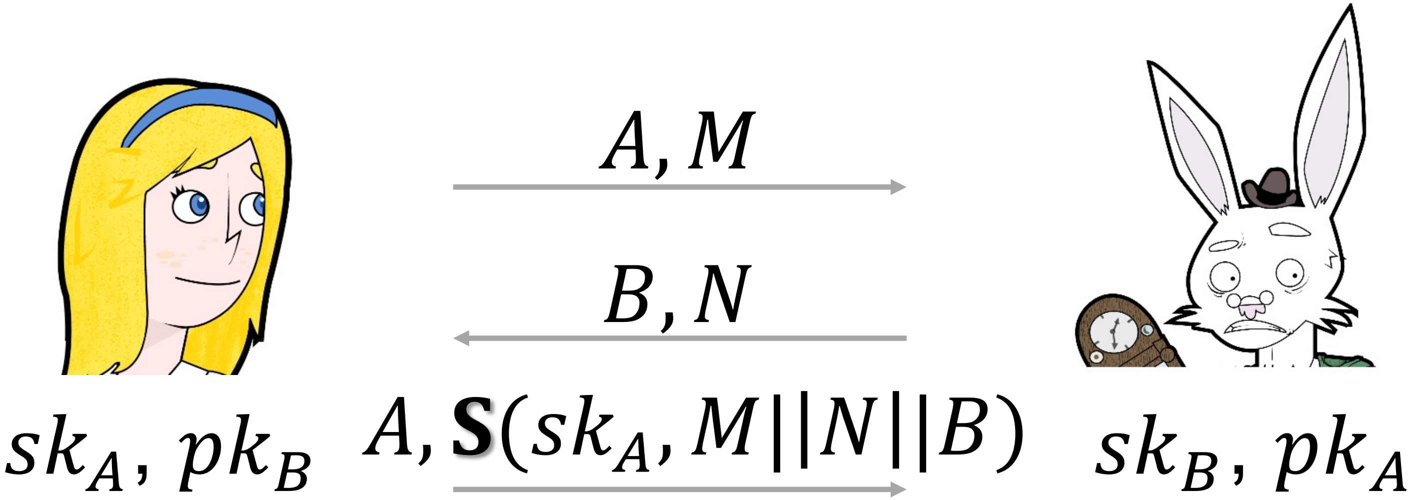
Sanity Checks

- The above definition is **simple** but **powerful**
  - **Impersonation:** If Eve can impersonate Bob **without** corrupting him, she **knows** a key for an **unexposed** session
  - Eve **can't** break one session given the key of **another** session
  - **Identity misbinding:** If Eve forces two (**non-matching**) sessions with outputs (A, B, K) and (B, E, K), she can choose one to be the **test session** and use the other one to **expose** K

Authenticators

- Consider a much **weaker** attack model where a KE protocol uses **authenticated** channels
  - **Idealized** model with **passive** attacker
  - Still the attacker can do **everything else**
  - The DH protocol is **trivially secure** in this model
- Authenticators are protocol **compilers** that allow to **reduce** KE protocols secure in the **unauthenticated** channels model to ones in the **authenticated** channels model

Authenticators based on Signatures



sk_A, pk_B

A, M →

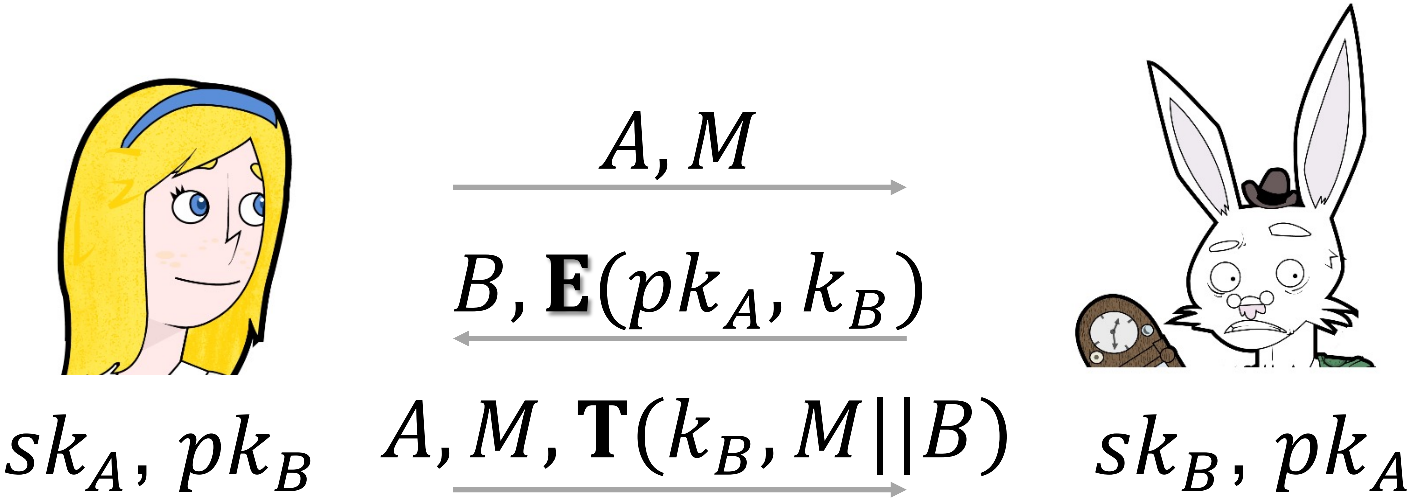
B, N ←

$A, \mathbf{S}(sk_A, M||N||B)$ →

sk_B, pk_A

- The nonce avoids **replay attacks**
- If Bob thinks that he **received** message M from Alice, then Alice **sent** M to Bob
  - One can show the above **implies** security of the ISO 9796 protocol in the CK model

Authenticators based on Encryption



A, M

$B, \mathbf{E}(pk_A, k_B)$

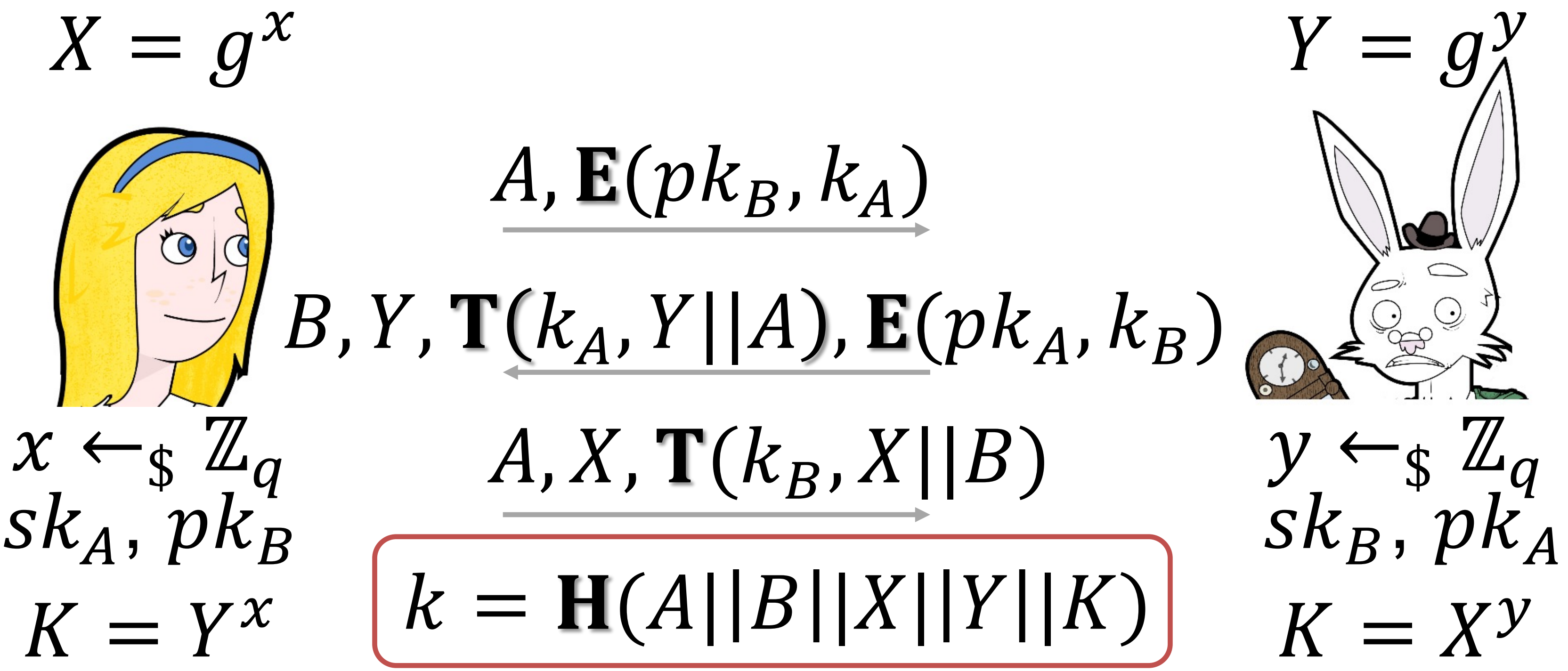
$A, M, \mathbf{T}(k_B, M||B)$

sk_A, pk_B

sk_B, pk_A

- Alice is the **only** party that can **decrypt** the ciphertext sent by Bob
  - Under **randomly chosen** key k_B
- So Bob is convinced it received M from Alice
  - The first message can actually be **dropped** here

SKEME (IKEv1)



$X = g^x$

$Y = g^y$

$$A, \mathbf{E}(pk_B, k_A) \longrightarrow$$

$$\longleftarrow B, Y, \mathbf{T}(k_A, Y||A), \mathbf{E}(pk_A, k_B)$$

$$A, X, \mathbf{T}(k_B, X||B) \longrightarrow$$

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

sk_A, pk_B

$K = Y^x$

$$\boxed{k = \mathbf{H}(A||B||X||Y||K)}$$

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

sk_B, pk_A

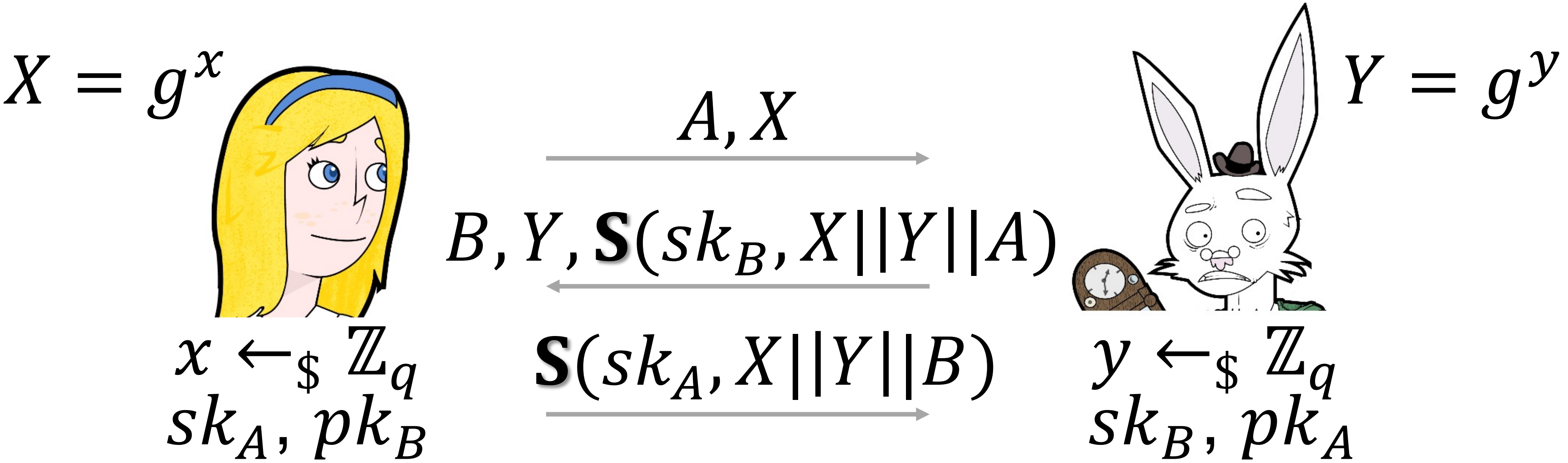
$K = X^y$

- The keys k_A and k_B are **randomly** chosen
- Can be seen as applying the **encryption-based** authenticator on the classical DH protocol

On Identity Protection

- **Identity** protection
  - Hide identities from **passive/active** adversaries
- A **privacy** concern in many scenarios
  - Probing attacks in the internet
  - Location anonimity of roaming users
- The design of IKE protocols in IPsec is **heavily influenced** by the above concern
  - SKEME and SIGMA
  - Typically **only one** id is hidden in the presence of active adversaries

Why not ISO?



$X = g^x$

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

sk_A, pk_B

$$\xrightarrow{A, X}$$

$$\xleftarrow{B, Y, \mathbf{S}(sk_B, X||Y||A)}$$

$$\xrightarrow{\mathbf{S}(sk_A, X||Y||B)}$$

$Y = g^y$

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

sk_B, pk_A

$K = Y^x$

$$\boxed{k = \mathbf{H}(A||B||X||Y||K)}$$

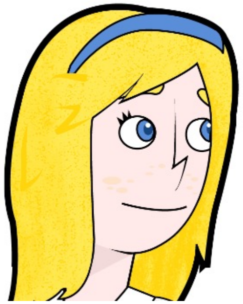
$K = X^y$

- **Unsuited** for identity protection
  - – Bob **needs to know** Alice’s identity and viceversa
  - – Also, it leaves a **signed proof** of communication

SKEME with Encrypted IDs

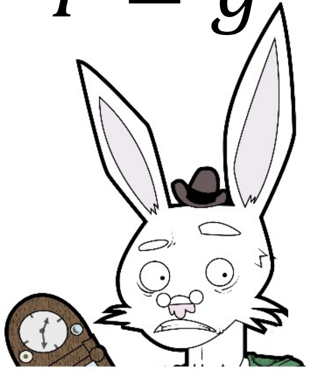
$X = g^x$

$Y = g^y$



$$\xrightarrow{\mathbf{E}(pk_B, A||k_A)}$$

$$\xleftarrow{Y, \mathbf{T}(k_A, Y||A), \mathbf{E}(pk_A, B||k_B)}$$



$$\xrightarrow{A, X, \mathbf{T}(k_B, X||B)}$$

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

sk_A, pk_B

$K = Y^x$

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

sk_B, pk_A

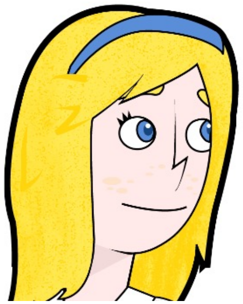
$K = X^y$

- The keys k_A and k_B are **randomly** chosen
- But Alice **needs to know** the public key of Bob **beforehand**

Alternative: Station-To-Station (STS)

$X = g^x$

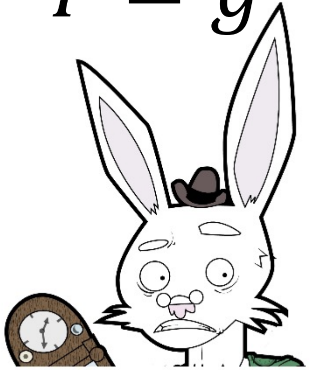
$Y = g^y$



$$\xrightarrow{X}$$

$$\xleftarrow{Y, \mathbf{E}(K, B||\mathbf{S}(sk_B, X||Y))}$$

$$\xrightarrow{\mathbf{E}(K, A||\mathbf{S}(sk_A, X||Y))}$$



$x \leftarrow_\$ \mathbb{Z}_q$
sk_A, pk_B

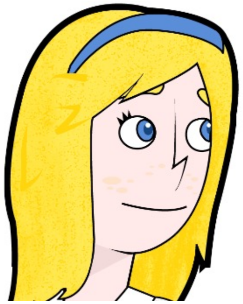
$K = Y^x$

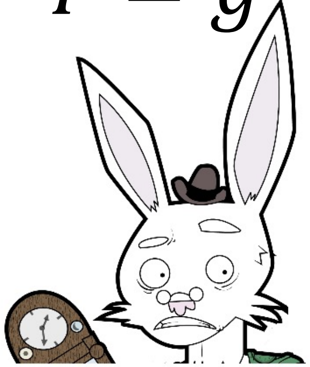
$y \leftarrow_\$ \mathbb{Z}_q$
sk_B, pk_A

$K = X^y$

- Add a **proof of knowledge** of the secret key K
- **Insecure** if Eve can **register** pk_A as her key
  - At least in the variant where A is **in the clear**

STS using MACs





$X = g^x$

$Y = g^y$

$$X \xrightarrow{\hspace{4cm}}$$

$$Y, B, \sigma_B = \mathbf{S}(sk_B, X||Y), \mathbf{T}(K, \sigma_B) \quad \xleftarrow{\hspace{4cm}}$$

$$A, \sigma_A = \mathbf{S}(sk_A, X||Y), \mathbf{T}(K, \sigma_A) \quad \xrightarrow{\hspace{4cm}}$$

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

sk_A, pk_B

$K = Y^x$

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

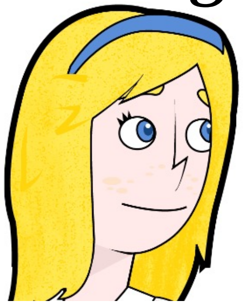
sk_B, pk_A

$K = X^y$

- MACs more suited to **prove knowledge** of K
- Yet, the **same attack** as before **still works**
  - – We need to **bind** the **key** with the **peer ids**

SIGMA: Basic Version

$X = g^x$



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

sk_A, pk_B

$K = Y^x$

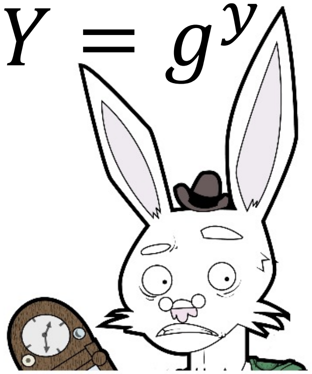
$$\xrightarrow{X}$$

$$\xleftarrow{B, Y, \mathbf{S}(sk_B, X||Y), \mathbf{T}(k', B)}$$

$$\xrightarrow{A, \mathbf{S}(sk_A, X||Y), \mathbf{T}(k', A)}$$

$$\boxed{k = \mathbf{H}(A||B||X||Y||K)}$$

$Y = g^y$



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

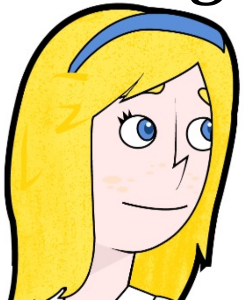
sk_B, pk_A

$K = X^y$

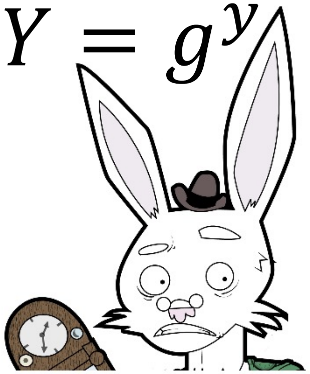
- Instead of signing Alice’s id (ISO), Bob **tags** its own identity with **another key** k'
  - The key k' is **derived** from K (as the session key k)

SIGMA-I: Protect Alice’s ID (Initiator)

$X = g^x$



$Y = g^y$



$$\xrightarrow{X}$$

$$\xleftarrow{Y, \mathbf{E}(k'', B||\mathbf{S}(sk_B, X||Y)||\mathbf{T}(k', B))}$$

$$\xrightarrow{\mathbf{E}(k'', A||\mathbf{S}(sk_A, X||Y)||\mathbf{T}(k', A))}$$

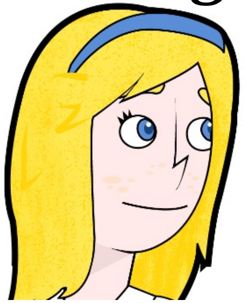
$x \leftarrow_\$ \mathbb{Z}_q$
sk_A, pk_B
$K = Y^x$

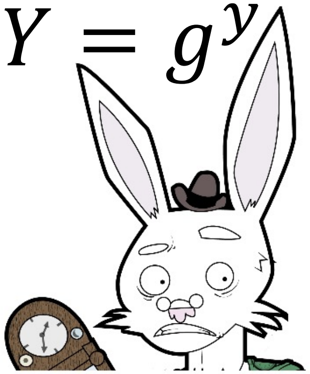
$y \leftarrow_\$ \mathbb{Z}_q$
sk_B, pk_A
$K = X^y$

$$\boxed{k = \mathbf{H}(A||B||X||Y||K)}$$

- **Encrypt** the identities of both Alice and Bob using **another key** k'' (still derived from k)
  - – Bob’s id is protected against **passive** attackers
  - – Alice’s id is protected against **active** attackers

SIGMA-R: Protect Bob's ID (Responder)





$X = g^x$

$Y = g^y$

$$X \xrightarrow{\hspace{4em}}$$

$$Y \xleftarrow{\hspace{4em}}$$

$$\mathbf{E}(k'', A||\mathbf{S}(sk_A, X||Y)||\mathbf{T}(k', A)) \longrightarrow$$

$$\mathbf{E}(k'', B||\mathbf{S}(sk_B, X||Y)||\mathbf{T}(k', B)) \longleftarrow$$

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

sk_A, pk_B

$K = Y^x$

$$\boxed{k = \mathbf{H}(A||B||X||Y||K)}$$

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

sk_B, pk_A

$K = X^y$

- Bob **does not** reveal his identity **before** checking who he is talking to
  - Bob's id is protected against **active** attackers
  - Alice's id is protected against **passive** attackers

Security of SIGMA

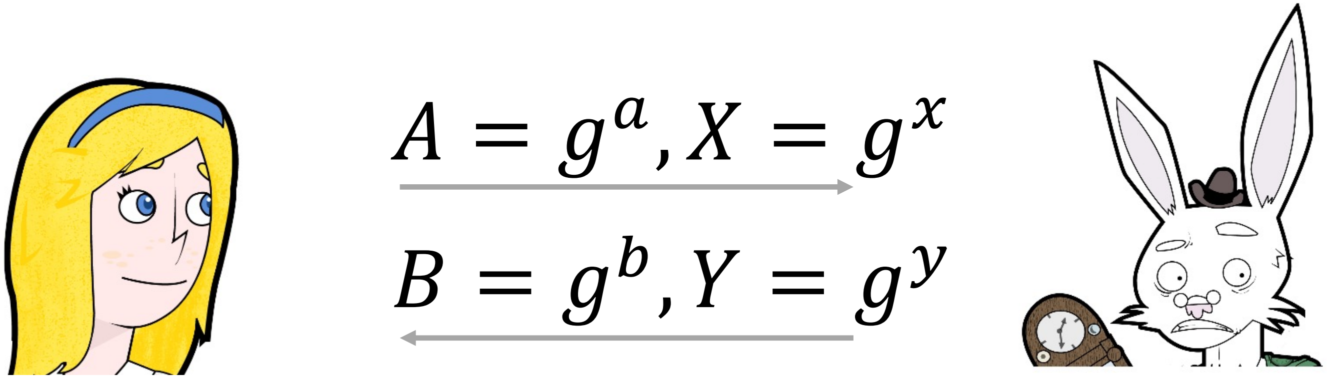
- The above description is **oversimplified** and glosses over a number of **details**
  - Additional information (context, negotiation, …)
- Nevertheless, SIGMA can be **proved** secure in the CK model
  - But no **modular** proof using **authenticators** is currently known
- The protocol is used in IPSec as well as part of the new TLS 1.3 standard

AKE with Implicit Authentication

- Drawbacks of the ISO 9796 protocol
  - It requires to send **signatures** and **certificates**
- What is the **inherent cost** of **authentication**?
  - Communication complexity
  - Computation complexity
  - What security?
- **Implicit** authentication
  - No signatures or tags sent
  - **Ability to compute** session key → authentication

Only the certificates are sent

Some Ideas

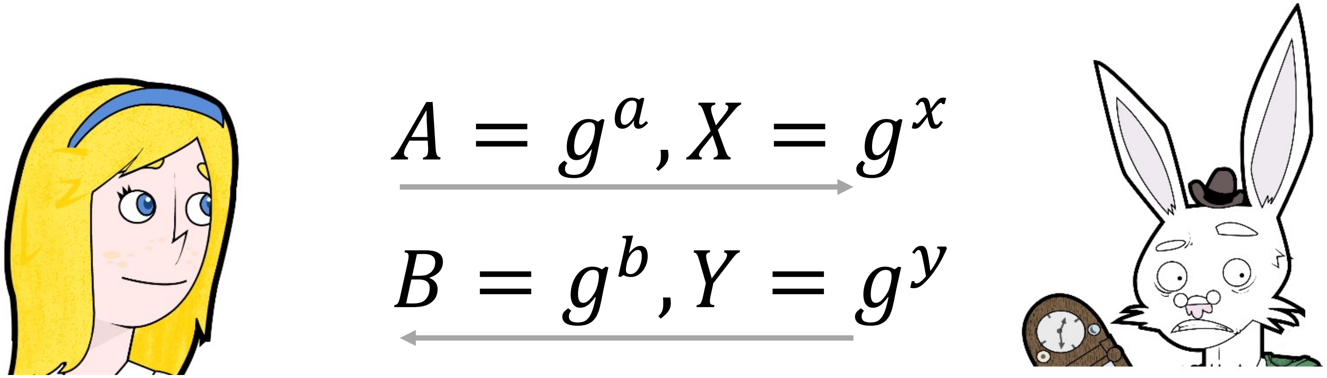


$$A = g^a, X = g^x$$

$$B = g^b, Y = g^y$$

- Many **insecure** attempts
  - $k = \mathbf{H}(g^{ab}, g^{xy})$: given a key g^{xy} for **one session** one can find a key for **another session**
  - $k = \mathbf{H}(g^{ab}, g^{xy}, g^x, g^y)$: knowing the key b of Bob one can **impersonate** Alice to Bob
- **Want:** security unless (a, x) or (b, y) **leak**

MQV: The Basic Idea

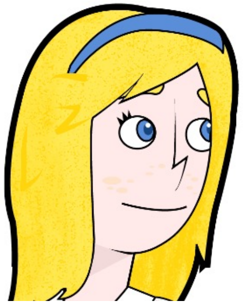


$$A = g^a, X = g^x \longrightarrow$$

$$B = g^b, Y = g^y \longleftarrow$$

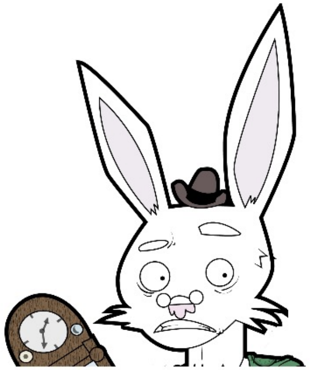
- **Idea:** Let $K = g^{(a+x)(b+y)}$
  - **Insecure:** Eve sends $X^* = g^{x^*}/A$; Bob sends Y, and thus $K = (BY)^{x^*}$ which is the same as computed by Bob $(AX^*)^{b+y} = (BY)^{x^*}$
- **Avoid** the attack by letting $K = g^{(x+ad)(y+be)}$
  - Values d, e s.t. Eve **can't** control e, Y or d, X

Hashed MQV



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

$$\xleftarrow{B = g^b, Y = g^y}$$



a, x

$$K = (YB^e)^{x+ad}$$

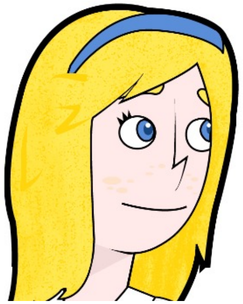
$$d = \mathbf{H}(X||\text{Bob})$$
$$e = \mathbf{H}(Y||\text{Alice})$$

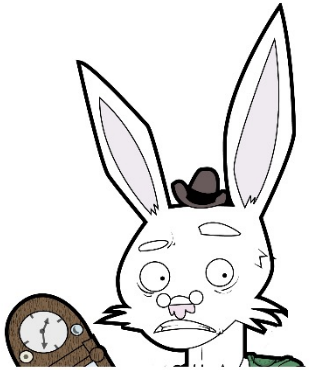
b, y

$$K = (XA^d)^{y+be}$$

- The **session key** is just $k = \mathbf{H}(K)$
  - Computing K requires $1 + 1/6$ **exponentiations**
- MQV: Let d be the **first half** bits of X and e be the **second half** bits of Y (but **insecure**)

Hashed MQV





$$A = g^{a}, X = g^{x} \xrightarrow{\hspace{4em}}$$

$$B = g^{b}, Y = g^{y} \xleftarrow{\hspace{4em}}$$

a, x

$K = (YB^{e})^{x+ad}$

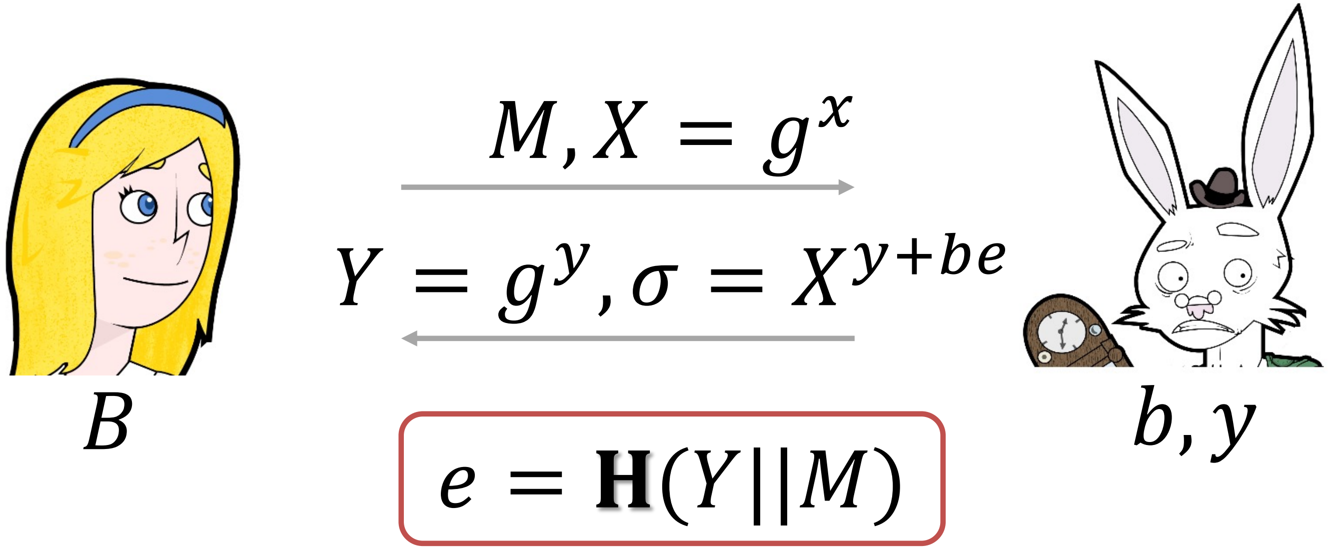
$$\boxed{\begin{array}{c} d = \mathbf{H}(X||\text{Bob}) \\ e = \mathbf{H}(Y||\text{Alice}) \end{array}}$$

b, y

$K = (XA^{d})^{y+be}$

- No signatures **exchanged**
  - But we can think of $(YB^{e})^{x+ad}$ (resp. $(XA^{d})^{y+be}$) as a **signature** of Alice on $X||\text{Bob}$ (resp. $Y||\text{Alice}$)
  - **Same** signature by **different** parties on **different** messages

XCR Signatures



B

$$M, X = g^x$$

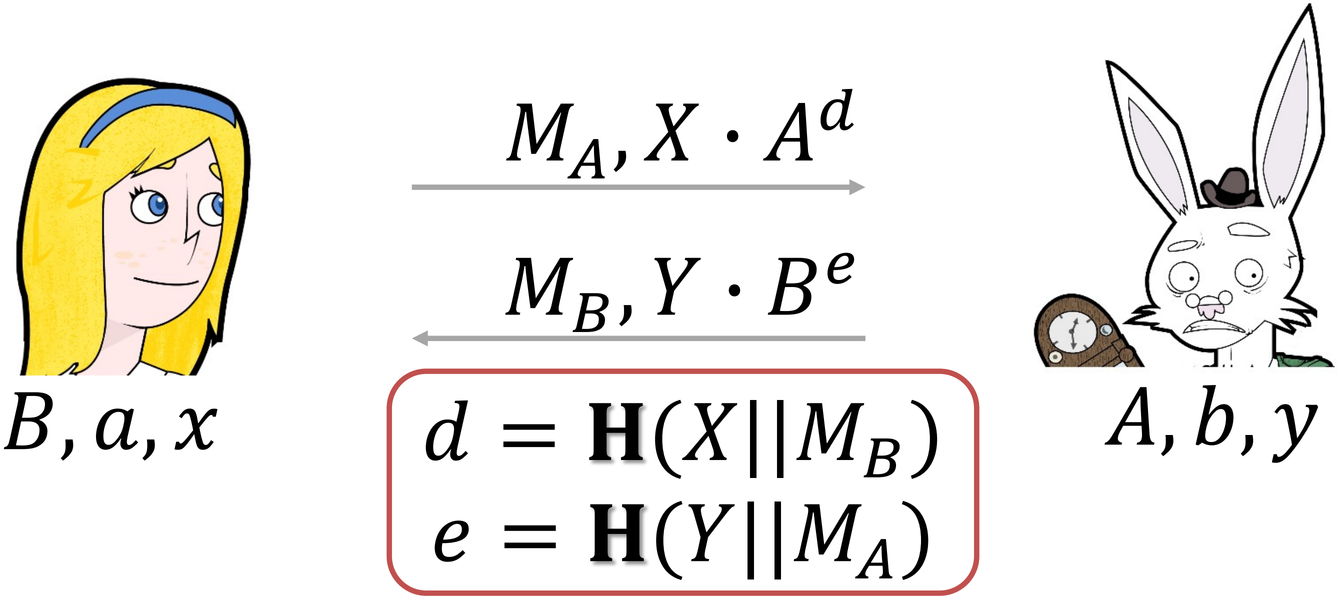
$$Y = g^y, \sigma = X^{y+be}$$

b, y

$$\boxed{e = \mathbf{H}(Y||M)}$$

- Bob is the **signer** with public key $B = g^b$
  - – Alice sends a **message** M and a **challenge** $X = g^x$
  - – Alice **accepts** iff $(YB^e)^x = \sigma$
- Alice is a **designated verifier**

Dual XCR Signatures



B, a, x

$M_A, X \cdot A^d$

$M_B, Y \cdot B^e$

A, b, y

$$d = \mathbf{H}(X || M_B)$$
$$e = \mathbf{H}(Y || M_A)$$

- Alice and Bob act as **simultaneous** signers
  - Bob (Alice) generates an **XCR signature** on **challenge** $X \cdot A^d$ $(Y \cdot B^e)$ and **message** M_A (M_B)
  - **Same** signature $\sigma = (XA^d)^{y+be} = (YB^e)^{x+ad}$

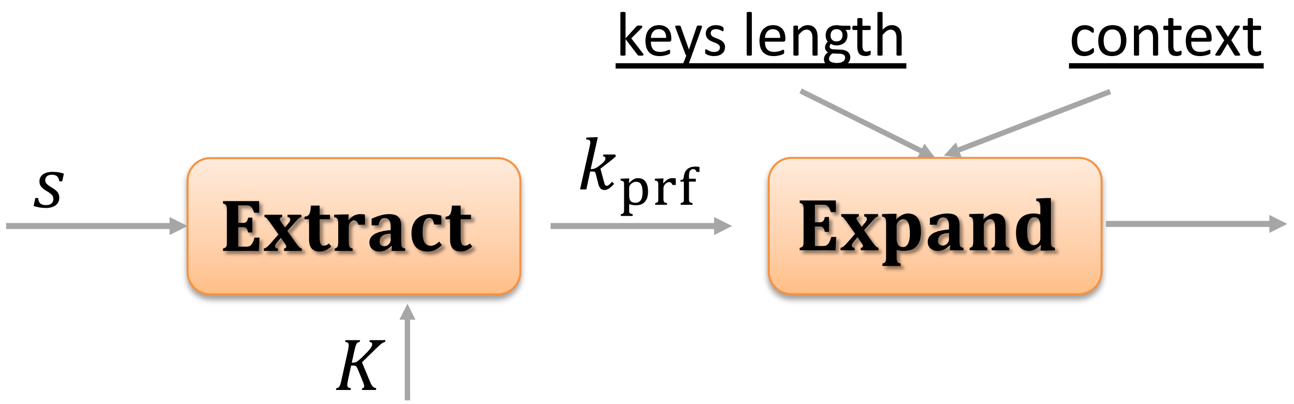
Security of HMQV

- One can show that HMQV is **secure** in the CK model (assuming **H** is a **random oracle**)
  - Reduce **security** of HMQV to **unforgeability** of Dual XCR signatures
  - Reduce **unforgeability** of Dual XCR signatures to **unforgeability** of XCR signatures
  - Reduce **unforgeability** of XCR signatures to the **CDH assumption** in the **random oracle model**
- The protocol is **standardized** by ANSI/ISO and IEEE, and also used by the NSA

Key Derivation Functions (KDFs)

- A KDF turns an **imperfect** source of randomness into one or more **random keys**
  - **Imperfect:** Not uniform
- In practice one just uses **random oracles**
  - As in $k = \mathbf{H}(g^{xy})$
  - Repeated extraction as $\mathbf{H}(g^{xy}||A)||\mathbf{H}(g^{xy}||B)$ …
- However, **no $\mathbf{H}$** can be a **random oracle**
  - **Length extension attack:** Given $\mathbf{H}(g^{xy}||A)$ can compute $\mathbf{H}(g^{xy}||B)$ if A is a **prefix** of B

Extract-than-Expand



keys length

context

s

Extract

k_{prf}

Expand

K

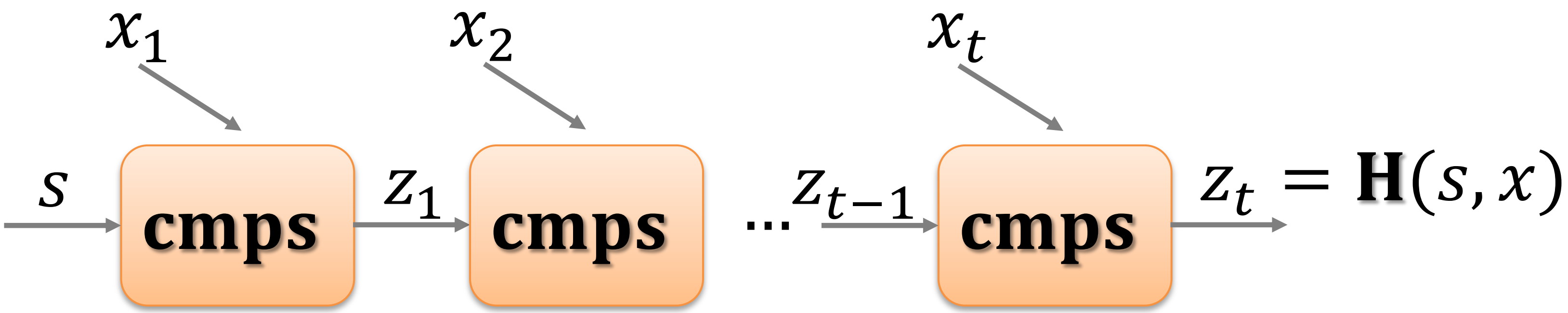
- The value s is a **salt** that is **public** but **random**
  - This is usually also **short**
- The value K is the starting **key material**
- Extract function: a **randomness extractor**
- Expand function: typically a **PRF**

Instantiations in Practice

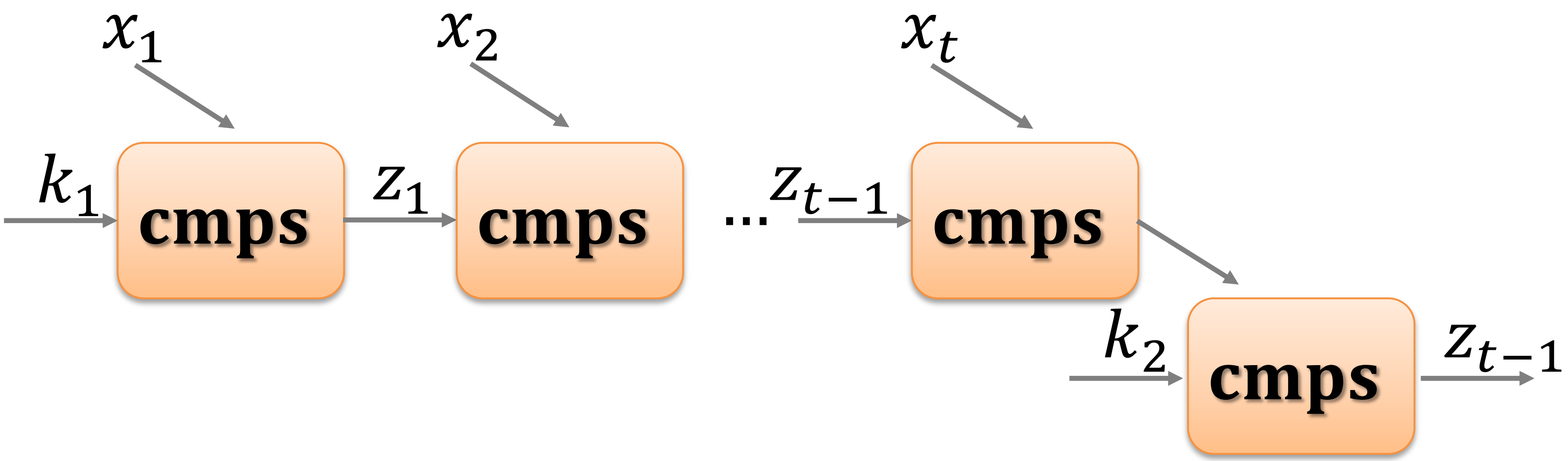
- There are **statistically-secure** extractors
  - But in **practice** those would require **large seeds** and yield quite **large entropy loss**
- **Alternative:** Use a PRF for **both** extraction and expansion
  - Difficulty: the seed is **public** (but the input is **not**)
  - There are **examples** of PRFs that **do not work**
- Luckily, the above works using **practical** PRFs
  - In particular, with the **standard** HMAC

Keyed Merkle-Damgaard

- Let **cmps** be a **compression function** outputting 160 bits out of 512 bits
- The **keyed** Merkle-Damgaard construction uses the seed s as **initial vector**

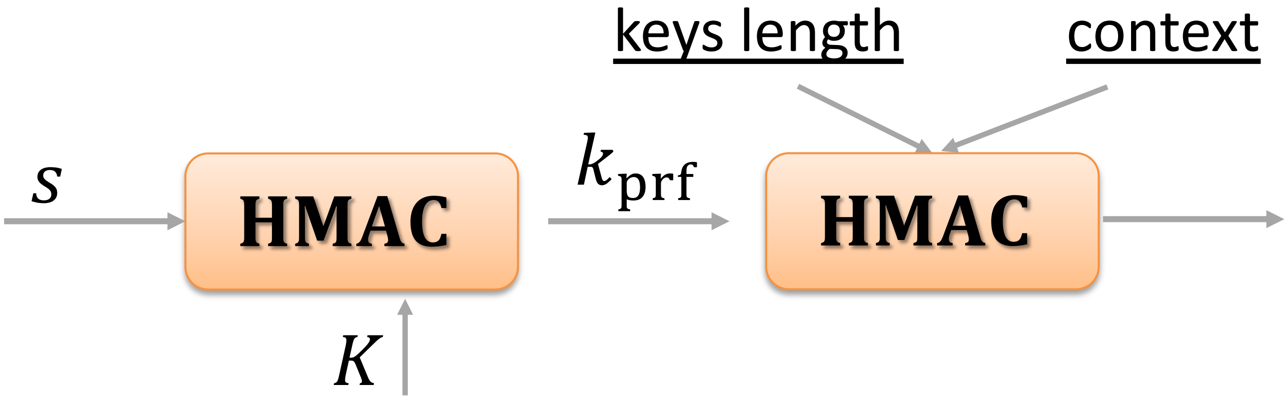


NMAC: PRF Mode for Merkle-Damgaard



- **Theorem:** $\mathbf{NMAC}(k_1||k_2,\cdot)$ is a PRF assuming $\mathbf{cmps}$ is a PRF
- HMAC is identical, but k_1, k_2 are **derived** from the **same key** k

Extract-than-Expand



- Expand function:

$$k_{i+1} = \mathbf{HMAC}(k_{\text{prf}}, k_i || \text{info} || i)$$

- This is HMAC as a PRF in **feedback mode**
- Heavily **standardized** (e.g., TLS 1.3, Whatsapp)
  - – And also **provably secure**

Applications of HKDF

- IPSec:
  - $k = \mathbf{HKDF}(\text{nonces}, g^{xy})$ where the **nonces** are part of the protocol and used as **salt**
  - In case the nonces are **public** the analysis requires that **HKDF** is an **extractor**
  - In case the nonces are **secret** (SKEME) the analysis requires that **HKDF** is a **PRF**
- TLS 1.3 with shared key $\hat{k}$ (**resumption**):
  - $k = \mathbf{HKDF}(\hat{k}, g^{xy})$
  - **HKDF** as an **extractor**/**PRF** if $\hat{k}$ is **revealed**/**secret**

Password-Authenticated Key Exchange

- **Authenticated** key exchange still requires a **public-key infrastructure**
- Alternative: Rely on a **shared password**
- The **standardization** of PAKE took several years starting back in 1982
- Today, PAKE is used in many **use cases**
  - TLS 1.3 (**pre-shared** key variant)
  - iCloud
  - RFID authentication

Passwords

- A **password** is a string of symbols belonging to a finite alphabet
  - Equivalently a bitstring
  - Needs to be **stored securely**
- Typical applications:
  - Derive a cryptographic key
  - Password-based authentication

Attacks on Passwords

- Guessing **always** possible (brute force)
  - **Online:** Trial & error
  - **Offline:** Dictionary attacks
- Sniffing from networks or theft from server
- Software attacks (trojan horse programs)
- Social engineering (phishing)
- Shoulder surfing

Online Password Guessing

- Always possible
  - Servers are always online
- Requires **interaction** with server
  - Limit number of **failed attempts**
  - Limit guessing rate
- Guessing rate
  - Attempt failure counter (but can’t block user account)
  - Increasing answer delay after each failed attempt

Offline Password Guessing

- Can’t be **detected**
- Attacker may choose **amount of resources**
- Complexity of guessing can be controlled by careful **password selection**
  - Given value $y = f(\pi, z)$, where f, z are public, a guessing attempt π' means to check $y = f(\pi', z)$

Passwords Entropy

- Let X be a random variable outputting symbols from an alphabet $\mathcal{A} = \{a_1, \dots, a_n\}$
- Denote by p_i the probability associated to a_i
- **Average information** in bit/symbol

$$H(X) = -\sum_{i=1}^{n} p_i \log p_i$$

- Maximum entropy for uniform distribution $H(U) = \log n$

ASCII Passwords

- Consider 7 bit ASCII: 95 **printable chars**
  - 0-31 are control chars
  - 127 is a special char
- For uniform passwords, with $n = 95$ we have $H(U) = \log 95 = 6.57$ bit/char
  - 128 bits of security correspond to random password of roughly 20 chars
- Situation **gets worse** if only upper/lower chars and numbers are used
  - $H(U) = \log 62 = 5.95$ bit/char

Passphrases

- More often users choose **passphrases**
- Let $p(\vec{x})$ be the probability of ℓ consecutive chars $\vec{x} = (x_1, \dots, x_\ell) \in \mathcal{A}^\ell$
- Now

$$H(X) = \lim_{\ell \to \infty} \frac{\overbrace{-\sum_{\vec{x} \in \mathcal{A}^\ell} p(\vec{x}) \log p(\vec{x})}^{H_\ell(X)}}{\ell}$$

- Italian language: $H_3(X) \approx 3.15$ bit/char; $H_5(X) \approx 2.22$ bit/char; $H_6(X) \approx 1.87$ bit/char

Users Choose Poor Passwords

- Study at Purdue University

Length	Number	Fraction of Total
1	55	0.4
2	87	0.6
3	212	2
4	449	3
5	1260	9
6	3035	22
7	2917	21
8	5772	42%

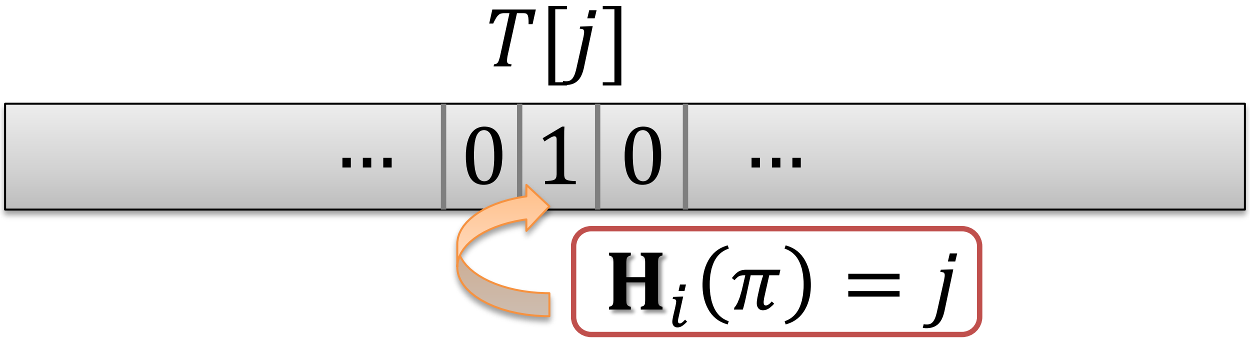
- Among 69 million Yahoo! Passwords, 1.1% of users pick **same password**

Password Selection

- Computer generated and **refreshed**
  - Difficult to remember!
- System process periodically tries **guessing** user passwords
  - CPU intensive
  - Memory intensive for big dictionaries
  - Users might get annoyed
- Check user password **as entered**
  - Simple guidance to select acceptable passwords

Bloom Filters (1/2)

- Tradeoff between accuracy and time/memory to check passwords **belong to dictionary** $\mathcal{D}$
- Let $\mathbf{H}_i$ be k hash functions yielding values in $[0, N-1]$ for $N = 2^s$ and T a table of N bits
- Let $y_i = \mathbf{H}_i(w), \forall w \in \mathcal{D}$ and set $T[y_i] = 1$
- Given π, reject it iff $T[\mathbf{H}_i(\pi)] = 1, \forall i \in [k]$



Bloom Filters (2/2)

- If $\pi \in \mathcal{D}$, it is **always rejected**
- If $\pi \notin \mathcal{D}$, it **might be rejected** (false positive)
  - Let $q = \Pr\left[T[j] = 0 : j \in [0, N-1]\right] = \Pr[\mathbf{H}_i(w) \neq j : \forall i \in [k], w \in \mathcal{D}]$
- False positive rate:

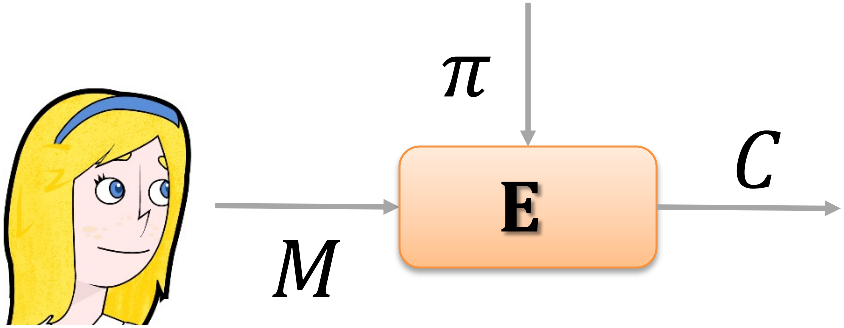
$$\boxed{p = (1-q)^k = (1-(1-1/N)^{kD})^k \approx (kD2^{-s})^k}$$

- Optimal values for fixed false positive rate:

$$k \approx -\log_2 p\,; N \approx -1.44 \cdot D \cdot \log_2 p$$

Password based Encryption

PKCS#5 Standard



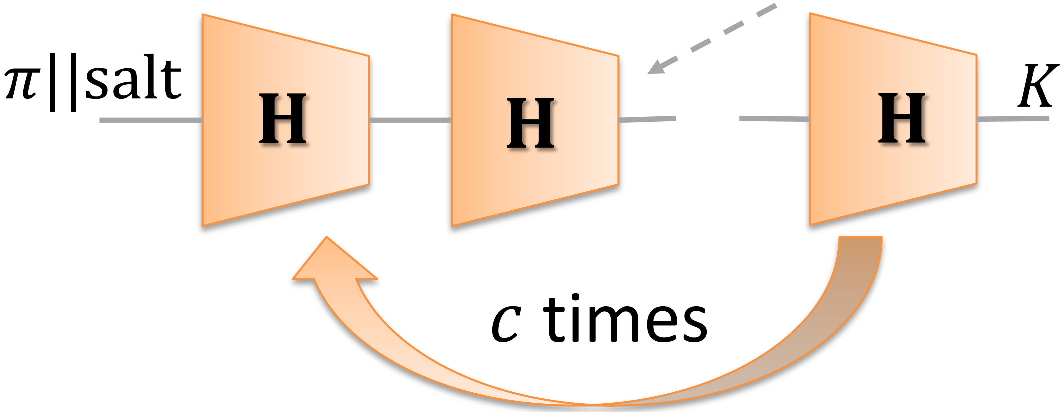
E(π, M):

salt $\leftarrow_{\$} \{0,1\}^{128}$

$K = \mathbf{H}^c(\pi||\text{salt})$

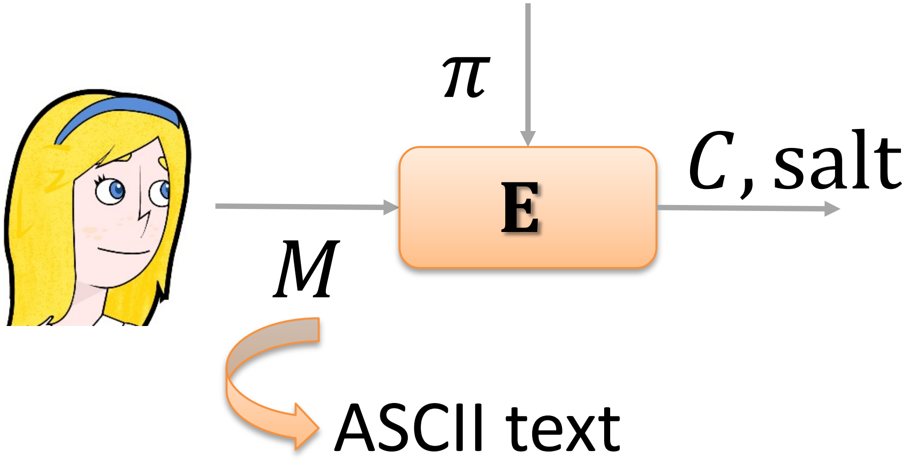
$C = K \oplus M$

Output (salt, C)



e.g., $c = 10000$

Salt and Stretching



- Hash chain **slows down** attacks by factor of c
- Salt defeats rainbow tables and provides **separation** between users

Typically assumed to be **trivial** for the adversary

C, salt

Step 1:

$M_1 = \mathbf{H}^c(\pi_1||\text{salt}) \oplus C$

$M_2 = \mathbf{H}^c(\pi_2||\text{salt}) \oplus C$

$M_3 = \mathbf{H}^c(\pi_3||\text{salt}) \oplus C$

…

Step 2:

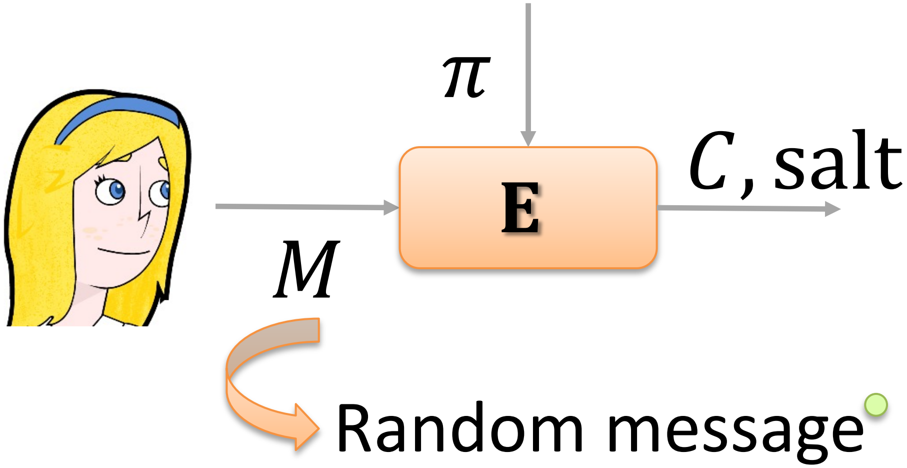
~~M_1 = as7e657q622!|a1~~

~~M_2 = mnas237@##saw~~

M_3 = sometext

…

Honey Encryption



Step 2 might be hard for some message distribution!

Seems **indistinguishable** to the adversary

C, salt

<u>Step 1:</u>

$M_1 = \mathbf{H}^c(\pi_1||\text{salt}) \oplus C$

$M_2 = \mathbf{H}^c(\pi_2||\text{salt}) \oplus C$

$M_3 = \mathbf{H}^c(\pi_3||\text{salt}) \oplus C$

$\ldots$

<u>Step 2:</u>

$M_1 = 01010000111000$

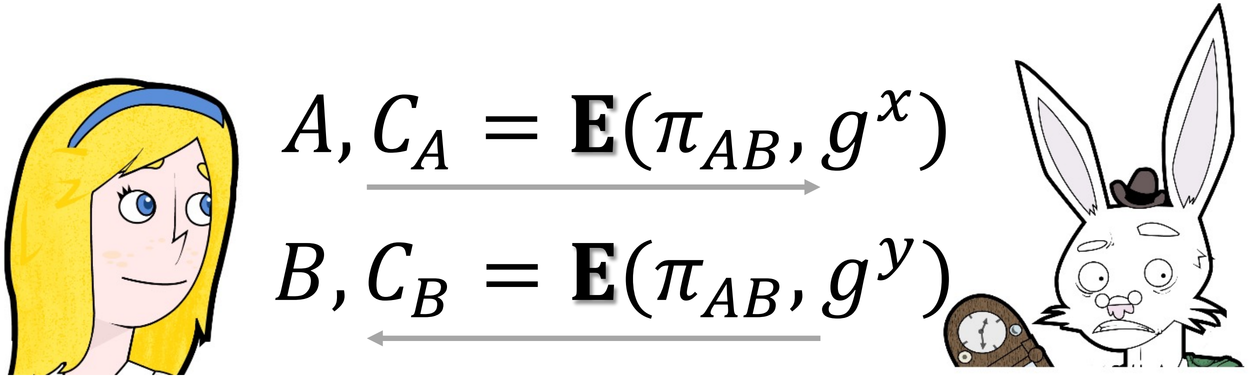
$M_2 = 01111100011000$

$M_3 = 11001111000101$

$\ldots$

Encrypted Key Exchange (EKE)

$k = \mathbf{D}(\pi_{AB}, C_B)^x$ $k = \mathbf{D}(\pi_{AB}, C_A)^y$



π_{AB}

$A, C_A = \mathbf{E}(\pi_{AB}, g^x) \xrightarrow{\hspace{2em}}$

$\xleftarrow{\hspace{2em}} B, C_B = \mathbf{E}(\pi_{AB}, g^y)$

π_{AB}

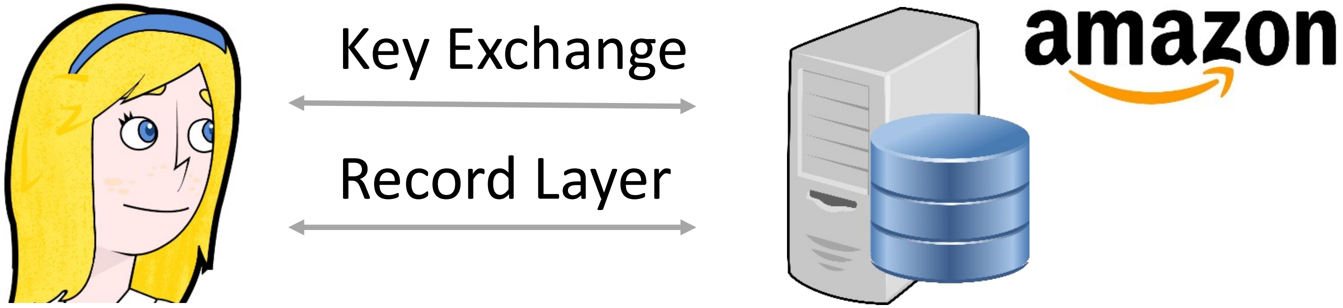
$x \leftarrow_\$ \mathbb{Z}_q$ $y \leftarrow_\$ \mathbb{Z}_q$

- Instantiation:
  - $\mathbf{E}(\pi, M) =$ **ideal cipher**
  - Hash protocol transcript with a **random oracle**

Transport Layer Security (TLS)

- Goal: Establish a **secure channel**
  - **Key exchange:** Yields keys for confidentiality/authenticity
  - **Record layer:** Use keys to secure communication
  - Authentication (usually on server side)
- Used in tons of applications
  - Amazon, ebay, e-commerce
  - Email
  - Google

The Client-Sever Scenario



- What actually happens:
  - – You type amazon.it in your browser
  - – TLS connection with Amazon is negotiated
  - – You get to https:// for **secure browsing**
  - – You **authenticate** to Amazon on a **secure link**

History of TLS

- Started out as Secure Socket Layer (SSL)
  - Developed by Netscape around 1995
  - Goal: Secure communication over Internet
- Changed to TLS in 1999
  - Secure communication (HTTPS)
  - … but also FTP, secure emailing, etc.
  - **Heavily standardized**
- Many implementations
  - OpenSSL, BoringSSL, s2n (TLS by Amazon)

SSL/TLS Versions

- SSL 1.0: Never released
  - Too **insecure** for release
- SSL 2.0: Released in February 1995
  - But contained a number of **security flaws**
- SSL 3.0: Released in 1996
- TLS 1.1: Protection against CBC-mode attacks
- TLS 1.2: Move from MD5 to SHA-1 (2008)
  - However, first attacks on MD5 **already in 2005**
- TLS 1.3: August 2018; completely **revised**

Attacks on TLS

- Renegotiation attack on SSL 3.0
  - **Ideal patch:** Kill renegotiation
  - **Real patch:** include previous session history
- Version rollback attacks
  - **Ideal patch:** Kill backward compatibility
  - **Real patch:** ??? (not a realistic attack)
- BEAST: Browser exploits of CBC vulnerabilities
  - **Ideal patch:** Kill CBC mode
  - **Real patch:** Discourage CBC mode

Attacks on TLS (cont'd)

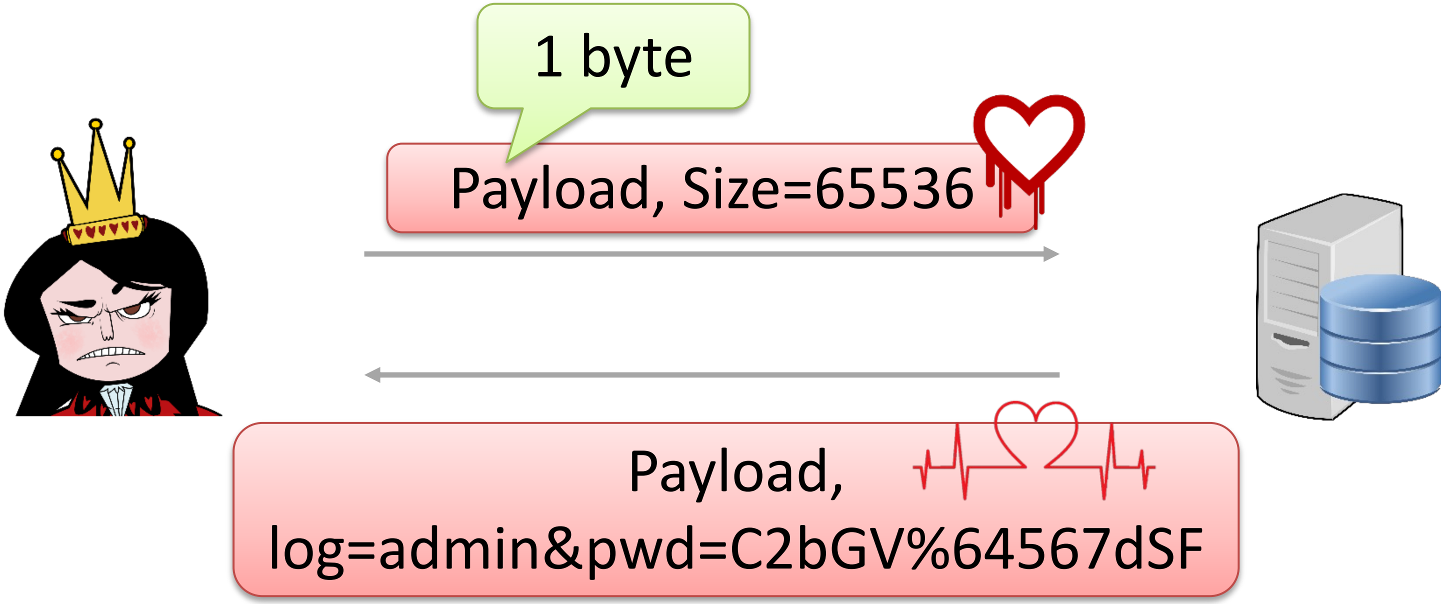
- Lucky 13: Exploit padding problems
  - **Ideal patch:** Kill CBC mode
  - **Real patch:** encouraged RC4 or use AES-GCM
- POODLE: Downgrade to SSL 3.0
  - **Ideal patch:** Kill backward compatibility
  - **Real patch:** ???

Even More Attacks

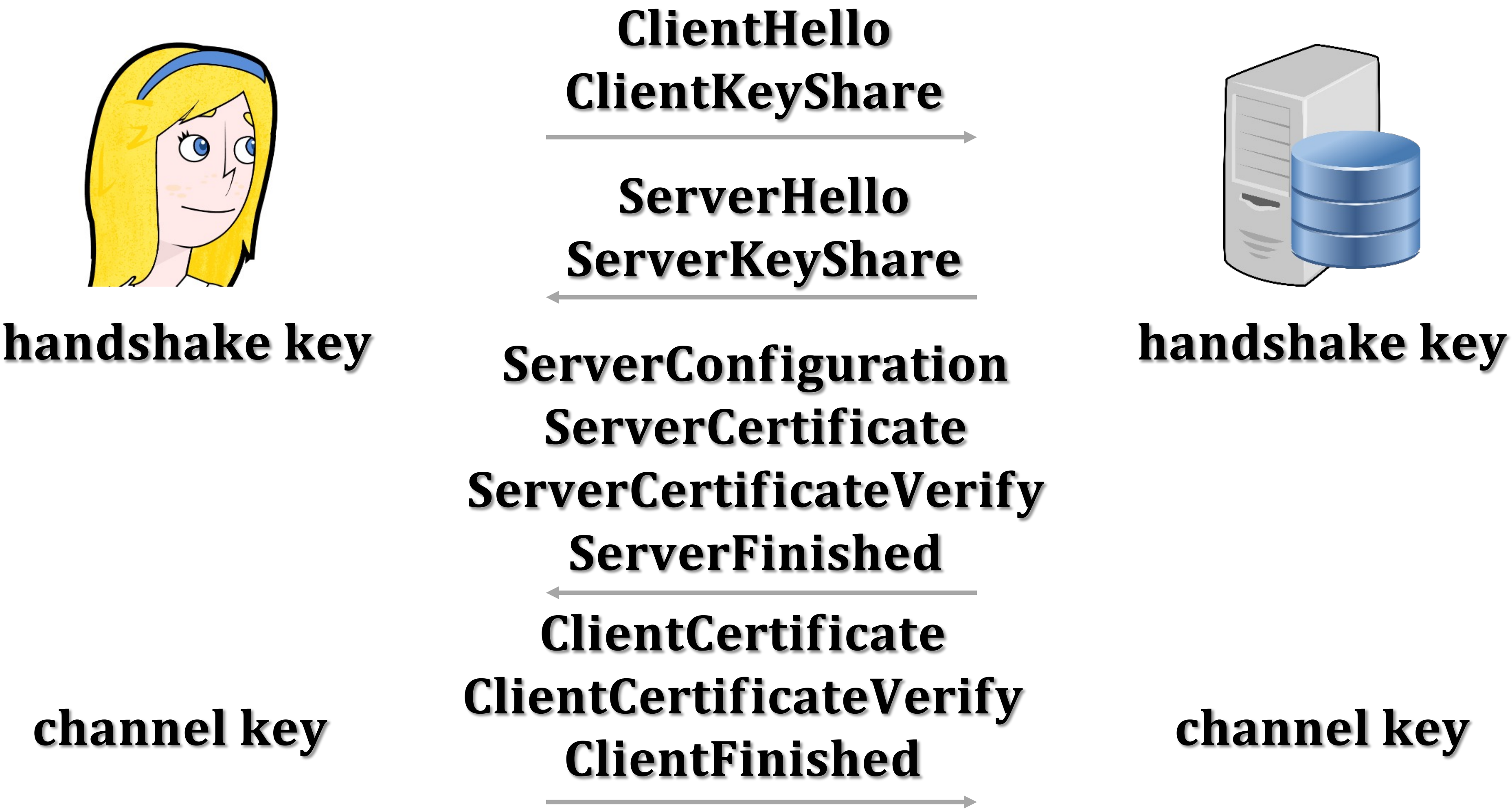
- RC4 attacks: RC4 output is biased
  - **Ideal patch:** Kill RC4
  - **Real patch:** RFC 7465 prohibits RC4, but
    - 30% of TLS traffic still uses RC4
    - 75% of sites allow RC4 negotiation
- Heartbleed, 3Shake, FREAK, Logjam
- …

Heartbleed

- Attack on OpenSSL based on **HeartBeats**
  - HeartBeat requests keep a TLS connection alive
  - HeartBeat contains a paylod along with its size



TLS 1.3: (EC)DHE



ClientHello
ClientKeyShare

ServerHello
ServerKeyShare

handshake key

ServerConfiguration
ServerCertificate
ServerCertificateVerify
ServerFinished

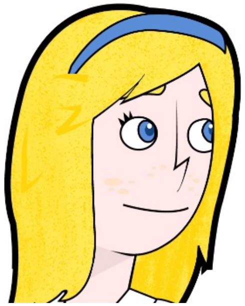
handshake key

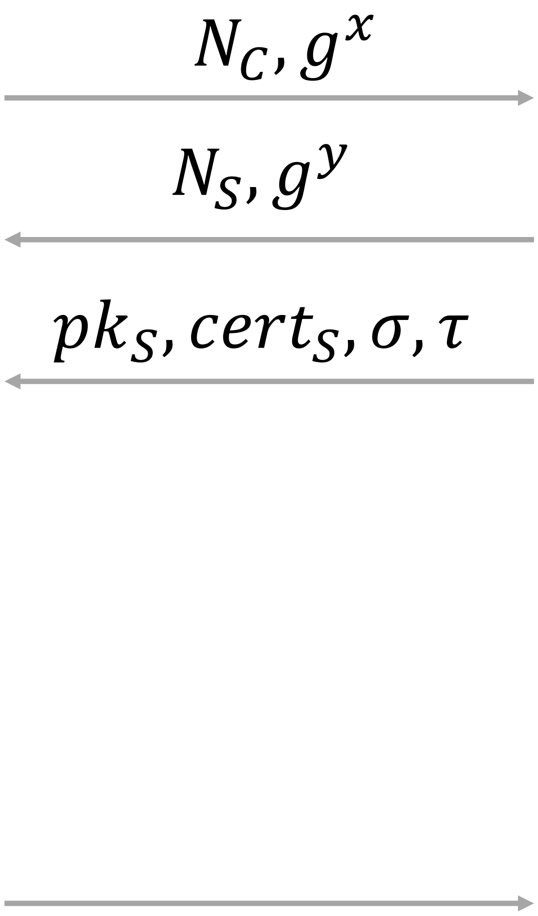
ClientCertificate
ClientCertificateVerify
ClientFinished

channel key

channel key

TLS 1.3: Crypto Details





$N_C \leftarrow \{0,1\}^{256}$

$x \leftarrow \mathbb{Z}_q$

handshake key

$\mathbf{KDF}(g^{xy}, CH, \dots, SKS)$

channel key

$\mathbf{KDF}(g^{xy}, CH, \dots, CF)$

$N_S \leftarrow \{0,1\}^{256}$

$y \leftarrow \mathbb{Z}_q$

handshake key

$\mathbf{KDF}(g^{xy}, CH, \dots, SKS)$

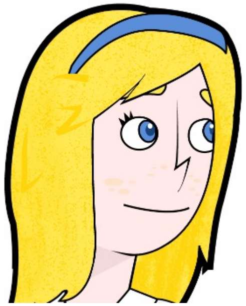
$\sigma = \mathbf{S}(sk_S, CH, \dots, SCert)$

$\tau = \mathbf{T}(k_{SF}, CH, \dots, SKS)$

channel key

$\mathbf{KDF}(g^{xy}, CH, \dots, CF)$

TLS 1.3: Pre-Shared Key Variant



ClientHello
ClientKeyShare
early_data
psk_ke_modes
psk_shared_key

→

preshared key

preshared key

ServerHello
ServerKeyShare
psk_shared_key
encrypted_extensions
ServerFinished

←

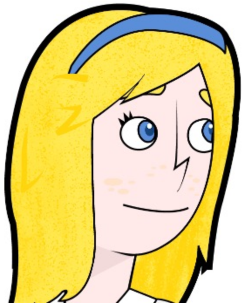
Externally or from session resumption

⋮

Zero Round-Trip Time

- TLS 1.3 requires **a few messages** before a key is established
- 0RTT is an alternative to the PSK variant
- The client starts the protocol and **immediately delivers** data
  - This is achieved using a **semi-static** server key
  - This key is available for **short** time periods
  - 0RTT was first invented by Google in order to reduce the latency

0RTT: QUIC



semi-static server key g^s

ephemeral key e, g^e

$k_1 = \mathbf{KDF}(g^{es})$

$\xrightarrow{g^e, \mathbf{E}(k_1, \text{data})}$

$\xleftarrow{\mathbf{E}(k_1, g^t)}$

semi-static server key s

$k_1 = \mathbf{KDF}(g^{es})$

ephemeral key t, g^t

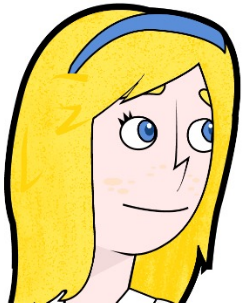
$k_2 = \mathbf{KDF}(g^{et})$

$\dashrightarrow$

$\xleftarrow{\mathbf{E}(k_2, \text{data})}$

$k_2 = \mathbf{KDF}(g^{et})$

Replay Attacks on QUIC



semi-static
server key g^s

ephemeral key e, g^e

$k_1 = \mathbf{KDF}(g^{es})$

$g^e, \mathbf{E}(k_1, \text{data})$ →

semi-static
server key s

$k_1 = \mathbf{KDF}(g^{es})$

$g^e, \mathbf{E}(k_1, \text{data})$ →

Only way out:
Store previously
received values