Computer and Network Security

Lecture 06:
KEX & Asymmetric Operations

COMP-5370/6370
Fall 2024

WARNING

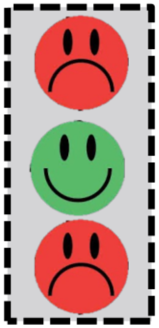
I AM NOT A CRYPTOGRAPHER

WARNING

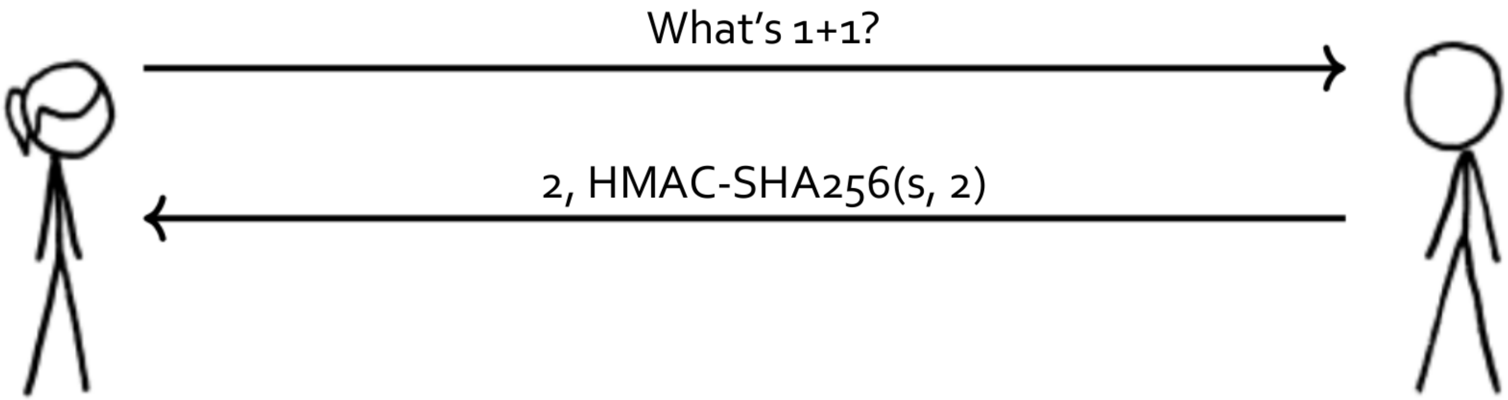
YOU ARE NOT A CRYPTOGRAPHER

- 4th Rule: Don’t roll your own crypto
- 5th Rule: Don’t roll your own crypto
- 6th Rule: Don’t roll your own crypto

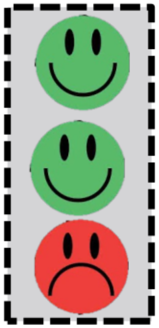
Building a Secure Channel



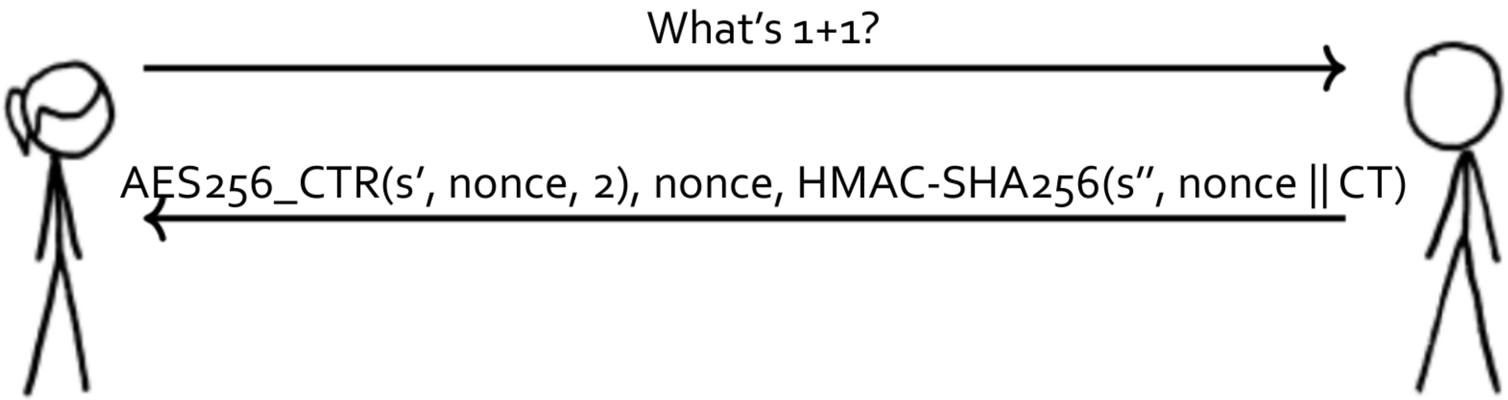
Confidentiality
Message Integrity
Sender Authenticity



Building a Secure Channel



Confidentiality
Message Integrity
Sender Authenticity

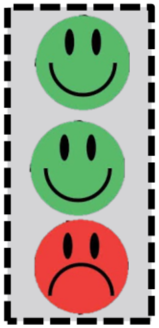


AEAD Cipher Modes

Authenticated Encryption with Associated Data (AEAD) cipher modes provide confidentiality and message integrity simultaneously.

- Provides **confidentiality**
- Provides **message integrity**
- Does **not** provide **sender authenticity**
- Vocab: `seal()` and `unseal()` instead of `encrypt()` and `decrypt()`

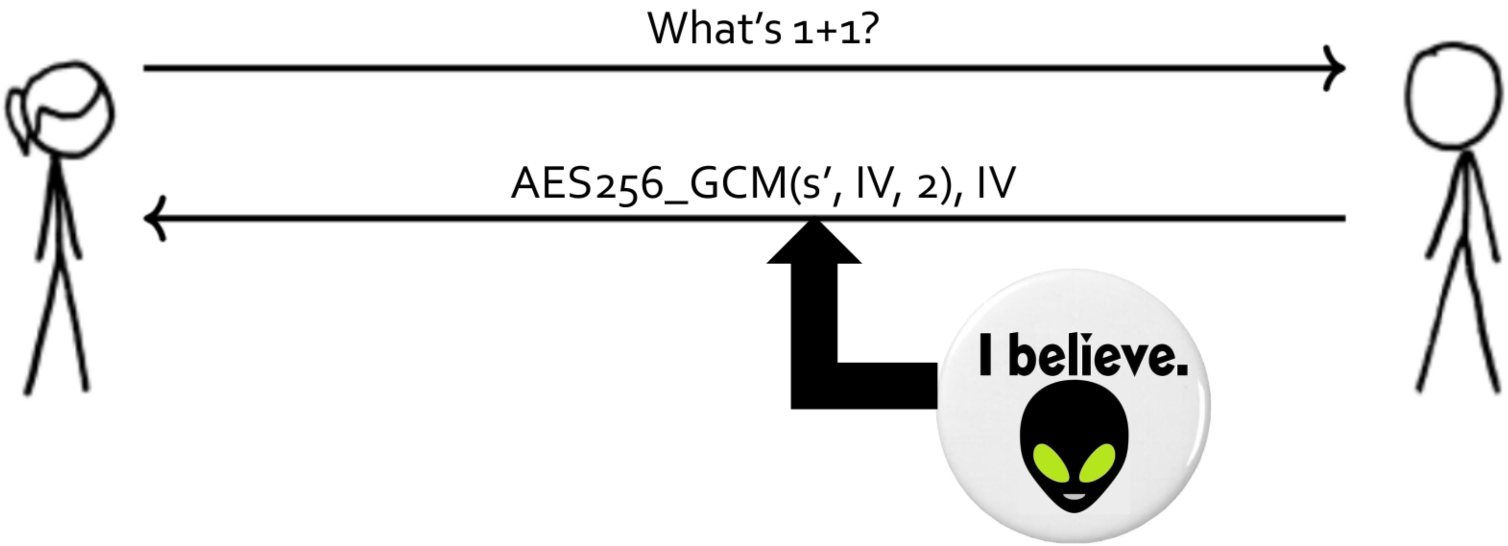
Building a Secure Channel



Confidentiality

Message Integrity

Sender Authenticity



Public Key Cryptography

Public key cryptography is a family of cryptosystems that leverage key pairs to perform asymmetric cryptographic operations.

asymmetric: Not a single shared secret between all parties

key pairs: Public key & Private key

- Public key == pub-key == pk
- Private key == priv-key == sk (“secret key”)

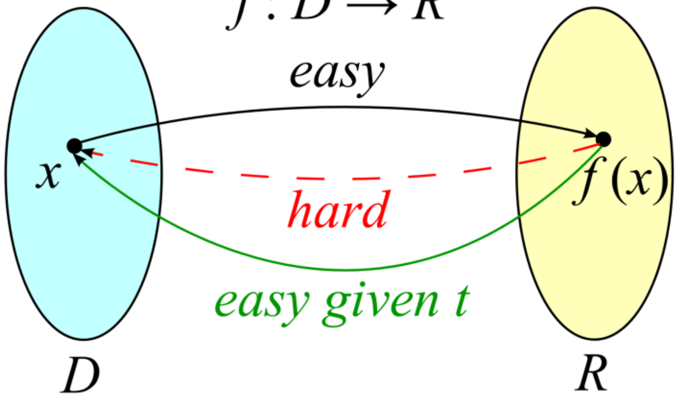
Trapdoor Function

A **trapdoor function** is one which can convert between two states but:

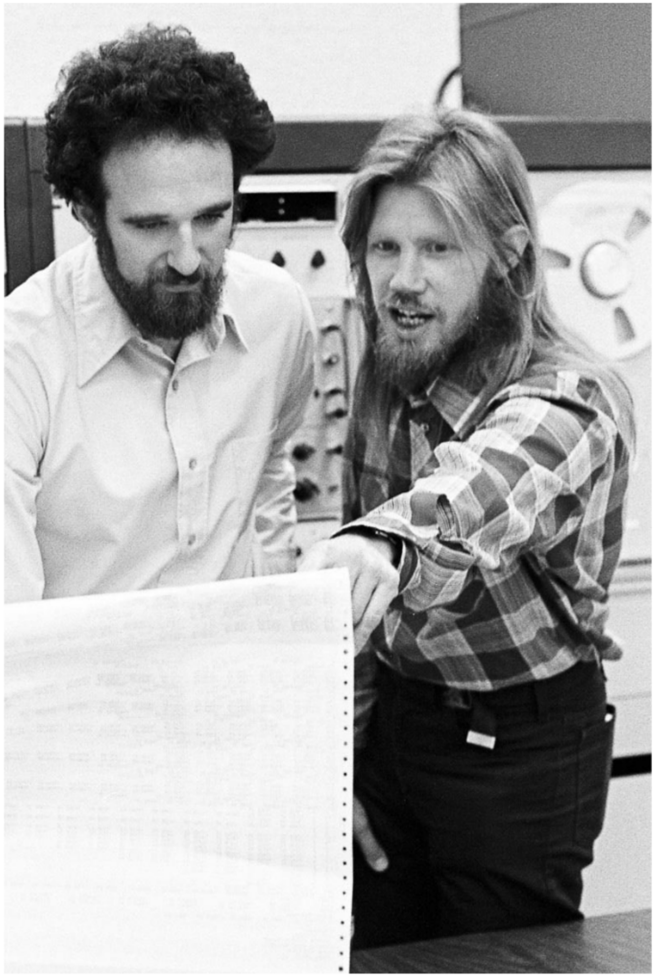
- Is computationally easy D → R
- Is computationally hard D ← R
- Is computationally easy D ← R given a secret

$(f, t) = \mathbf{Gen}(1^n)$

$f : D \to R$



Diffie-Hellman Key Exchange



- 1976 – Whit Diffie & Martin Hellman
  - *New Directions in Cryptography*
- Modular Exponentiation w/ Prime Modulus
  - *If you multiply a value by itself enough times over a prime-order finite field … you can't figure out how many times you multiplied*

Modular Exponentiation

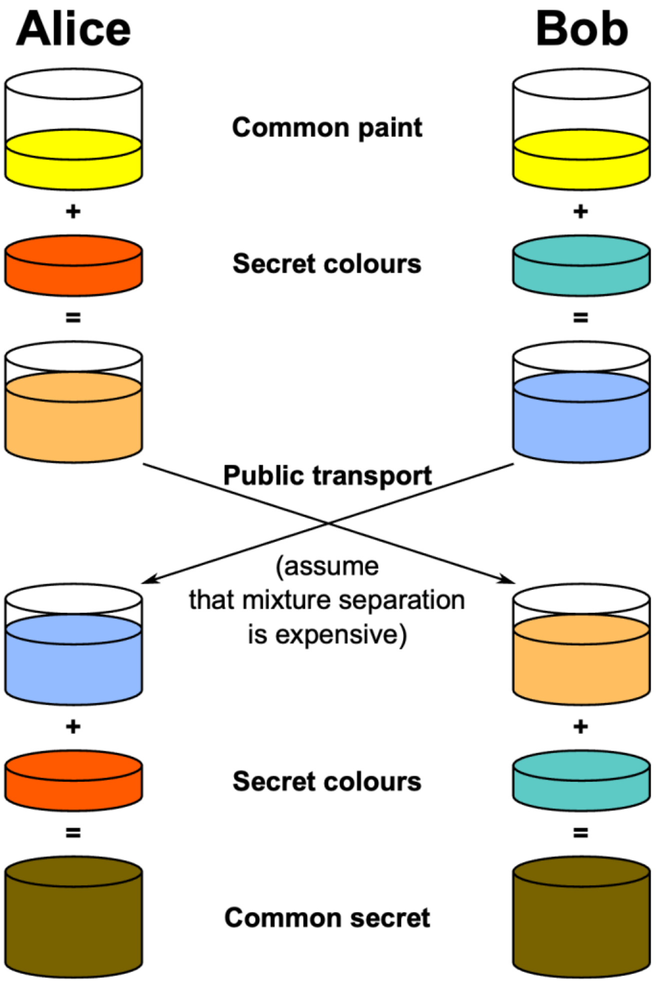
$$A^B \bmod C$$

$$2^2 \% 7 = 4 \% 7 = 4$$

$$2^3 \% 7 = 8 \% 7 = 1$$

$$2^8 \% 7 = 256 \% 7 = 4$$

Diffie-Hellman Key Exchange

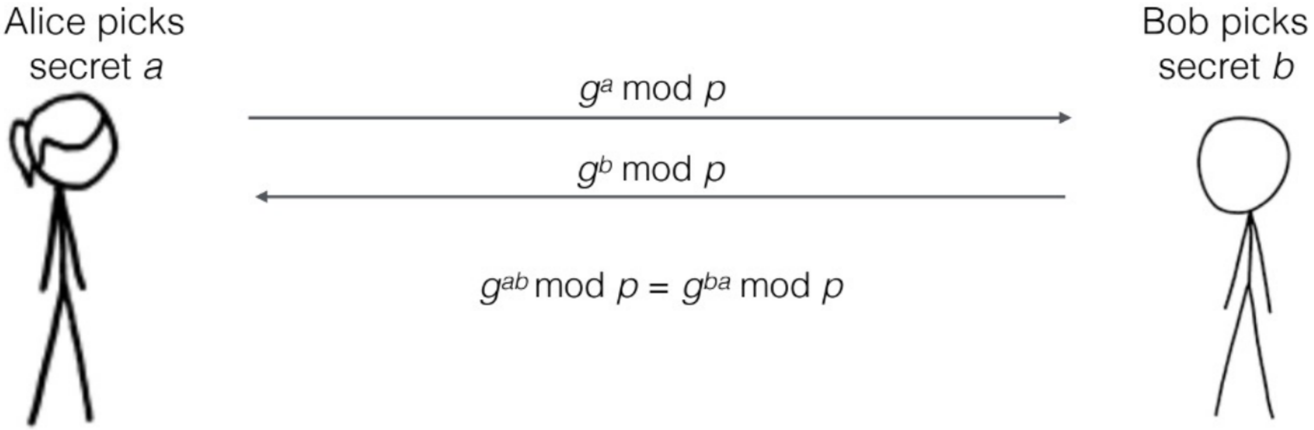


The **Diffie-Hellman Key Exchange** is a construction through which two parties can safely create a shared secret in the presence of a passive attacker.

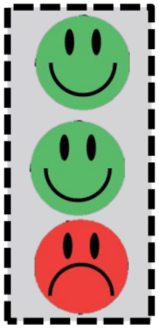
- *Many names:*
  - *DH Key Exchange*
  - *DH KEX*
  - *Ephemeral DH*
  - *DHE*

How Finite-Field DH Works

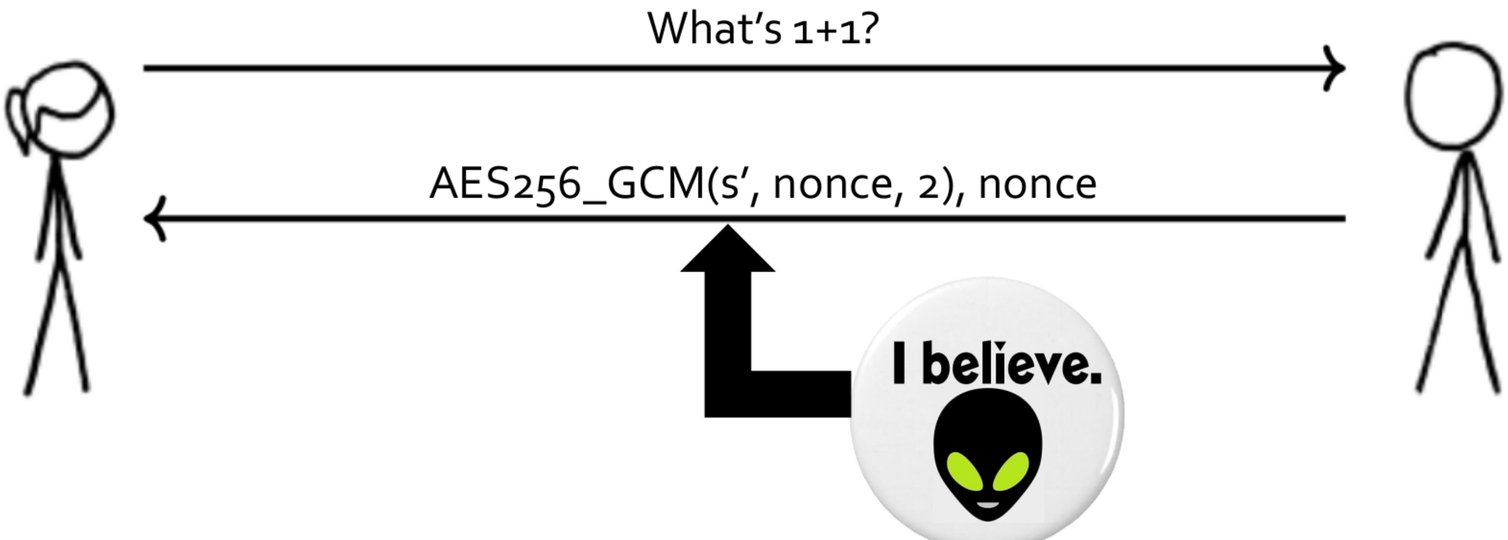
- Actors share generator *g* and prime modulus *p*
- Each actor selects a secret (*a*, *b*)
- Create/Send “keyshares” based on secret
- Use own secret and others’ keyshare to create a shared secret only known to actors



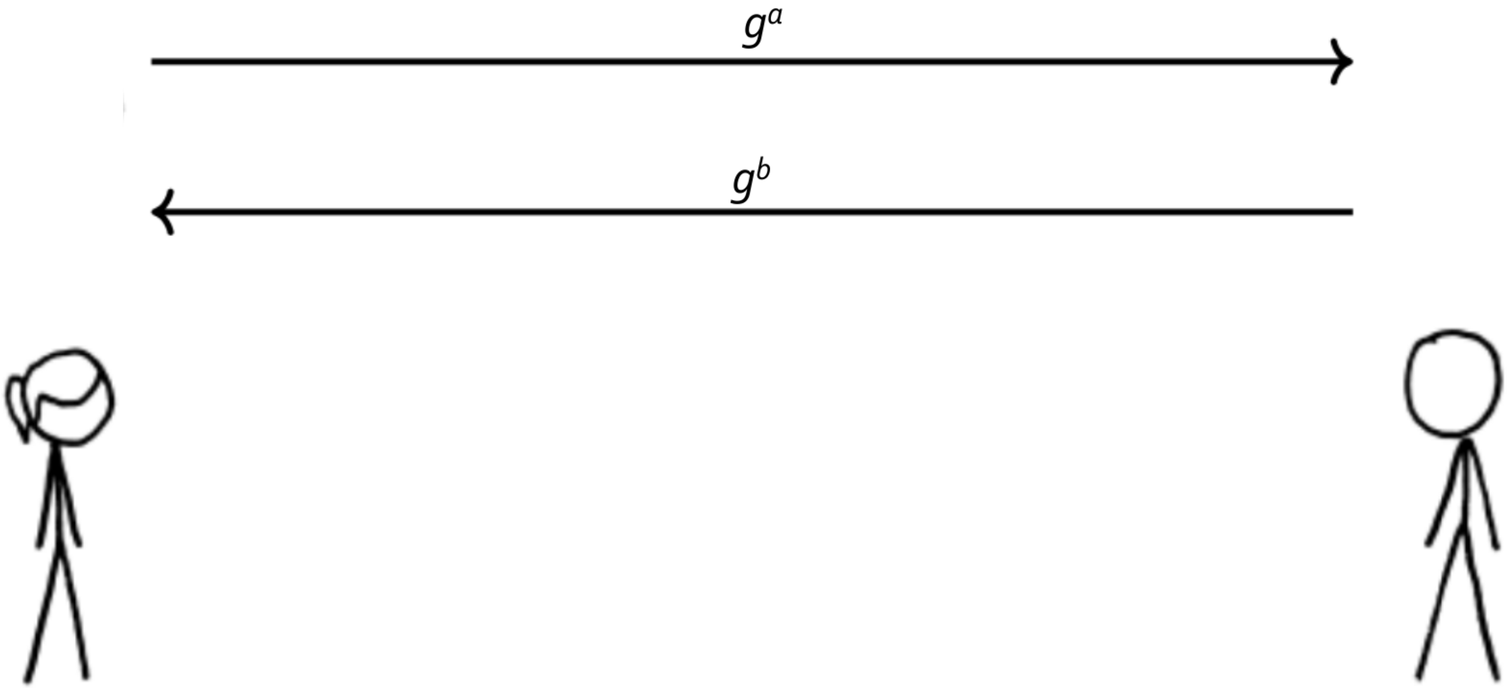
Building a Secure Channel



- Confidentiality
- Message Integrity
- Sender Authenticity



Building a Secure Channel



Security of Finite-Field DH

DH's security is based on the **assumed** hardness of two mathematical problems:

Discrete Logarithm Problem	Decisional Discrete Logarithm Problem
Given $g^x \bmod p$, it is hard to find x	Given $g^a \bmod p$ and $g^b \bmod p$, it is hard to find $g^{ab} \bmod p$

Safe Finite-Field DH

- Correctly generated 2048-bit group
  - Thought to be safe
  - Widely used in the real-world

- Correctly generated 3072-bit group
  - Thought to be safe
  - Relatively rare in the real-world
  - CNSA approved

Canonical DH Vulnerabilities

- Poor randomness when selecting *a* or *b*
  - If can recover one, g^{ab} *mod p* is trivial
- Poor selection of parameters
  - Pohlig-Hellman Algorithm
  - Non-trivial sub-group with different generator

Modular Exponentiation using Finite-Fields

$$A^B \bmod C$$

$2^2 \% 7 = 4 \% 7 = 4$

$2^3 \% 7 = 8 \% 7 = 1$

$2^8 \% 7 = 256 \% 7 = 4$

Modular Exponentiation using Finite-Fields

$$A^B \bmod C$$

`A == 2; C == 7`

$2^2 \% 7 = 4 \% 7 = 4$

$2^3 \% 7 = 8 \% 7 = 1$

$2^8 \% 7 = 256 \% 7 = 4$

Modular Exponentiation using Finite-Fields

$A^B \bmod C$

What happens with different values for C?

Modular Exponentiation using Finite-Fields

$$A^B \bmod C$$

<u>C == 6</u>

2^1 % 6 == 2
2^2 % 6 == 4
2^3 % 6 == 2
2^4 % 6 == 4
2^5 % 6 == 2

<u>C == 8</u>

2^1 % 8 == 2
2^2 % 8 == 4
2^3 % 8 == 0
2^4 % 8 == 0
2^5 % 8 == 0
2^6 % 8 == 0
2^7 % 8 == 0

Modular Exponentiation using Finite-Fields

$A^B \bmod C$

What happens with different values for A?

Modular Exponentiation using Finite-Fields

$$A^B \bmod C$$

A == 3	A == 4	A == 5
3^1 % 7 == 3	4^1 % 7 == 4	5^1 % 7 == 5
3^2 % 7 == 2	4^2 % 7 == 2	5^2 % 7 == 4
3^3 % 7 == 6	4^3 % 7 == 1	5^3 % 7 == 6
3^4 % 7 == 4	4^4 % 7 == 4	5^4 % 7 == 2
3^5 % 7 == 5	4^5 % 7 == 2	5^5 % 7 == 3
3^6 % 7 == 1	4^6 % 7 == 1	5^6 % 7 == 1
3^7 % 7 == 3	4^7 % 7 == 4	5^7 % 7 == 5

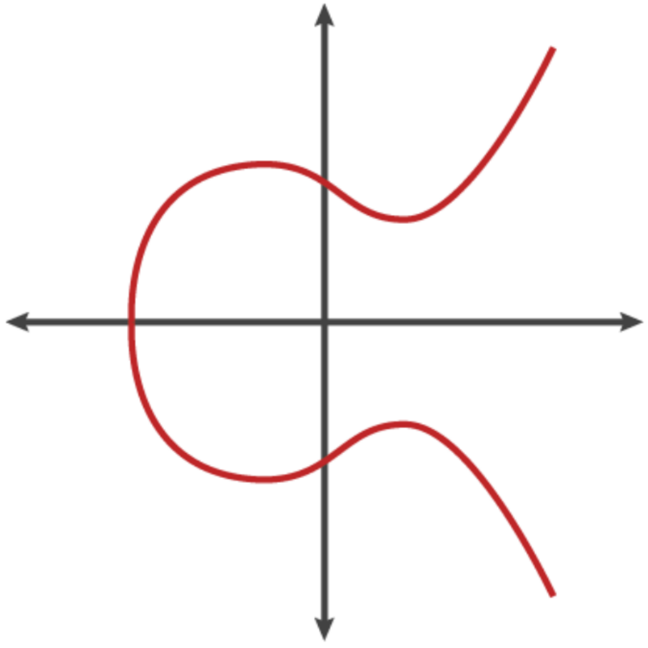
Canonical DH Vulnerabilities

- Poor randomness when selecting *a* or *b*
  - If can recover one, g^{ab} *mod p* is trivial
- Poor selection of parameters
  - Pohlig-Hellman Algorithm
  - Non-trivial sub-group with different generator
- Computation over-match
  - Discrete Log Record: 795-bit in ~100 days
- Break the group (defined by *g* and *p*)
  - *We'll talk about this later* :-)

Elliptic Curves over Finite-Fields

$$y^2 = x^3 + ax + b$$

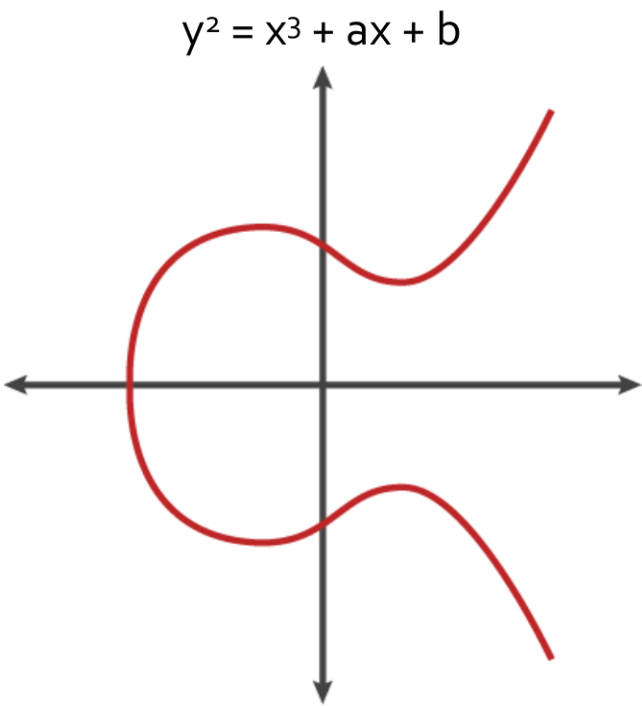
$$y = SQRT(x^3 + ax + b)$$



Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is a public-key scheme that provides the same operations via a different mechanism.

- Operates on elliptic curves over prime-order finite-fields



Real Cryptography

Theorem 19.18. *The AND protocol (P, V) is a Sigma protocol for the relation $\mathcal{R}_{\text{AND}}$ defined in (19.22). If (P_0, V_0) and (P_1, V_1) provide knowledge soundness, then so does (P, V). If (P_0, V_0) and (P_1, V_1) are special HVZK, then so is (P, V).*

Proof sketch. Correctness is clear.

For knowledge soundness, if (P_0, V_0) has extractor Ext_0 and (P_1, V_1) has extractor Ext_1, then the extractor for (P, V) is

$$Ext\Big((y_0, y_1), ((t_0, t_1), c, (z_0, z_1)),\ ((t_0, t_1), c', (z_0', z_1')) \Big) :=$$
$$\Big(Ext_0(y_0, (t_0, c, z_0), (t_0, c', z_0')),\ Ext_1(y_1, (t_1, c, z_1), (t_1, c', z_1')) \Big).$$

For special HVZK, if (P_0, V_0) has simulator Sim_0 and (P_1, V_1) has simulator Sim_1, then the simulator for (P, V) is

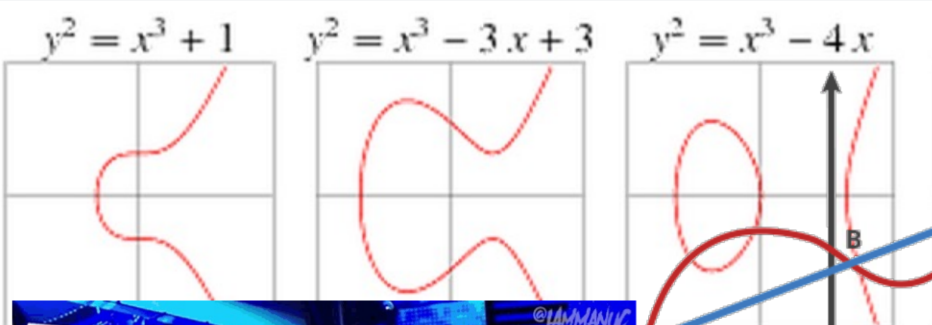
$$Sim((y_0, y_1), c) := ((t_0, t_1), (z_0, z_1)),$$

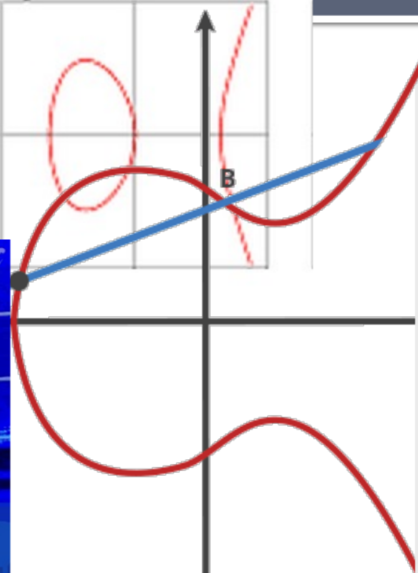
where

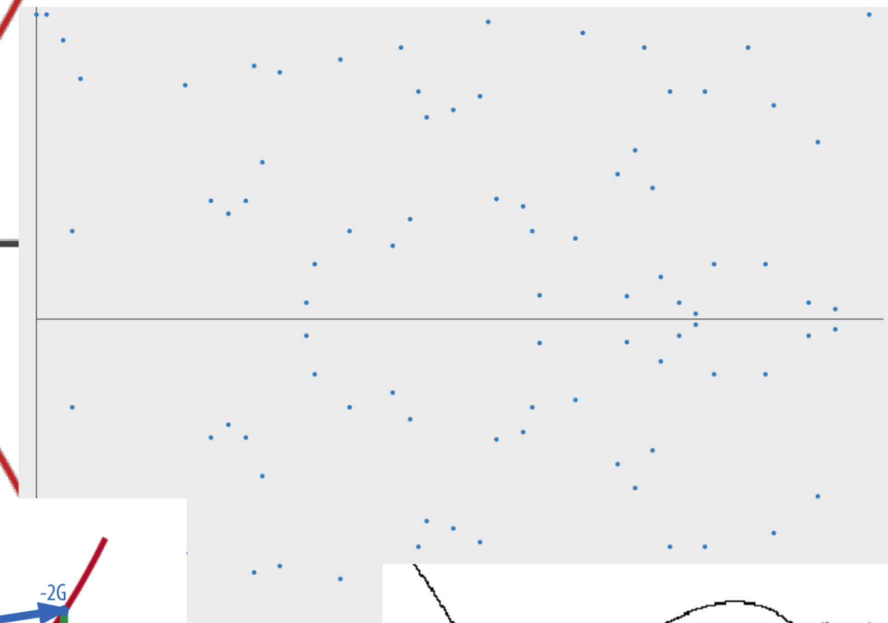
$$(t_0, z_0) \xleftarrow{R} Sim_0(y_0, c) \quad \text{and} \quad (t_1, z_1) \xleftarrow{R} Sim_1(y_1, c).$$

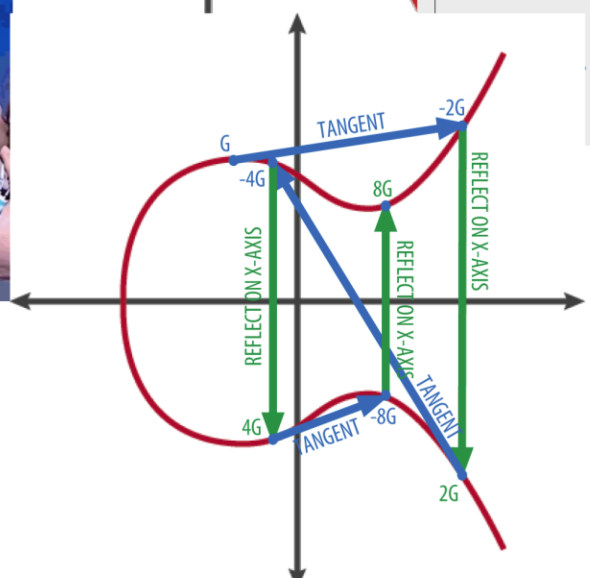
NOT THIS COURSE

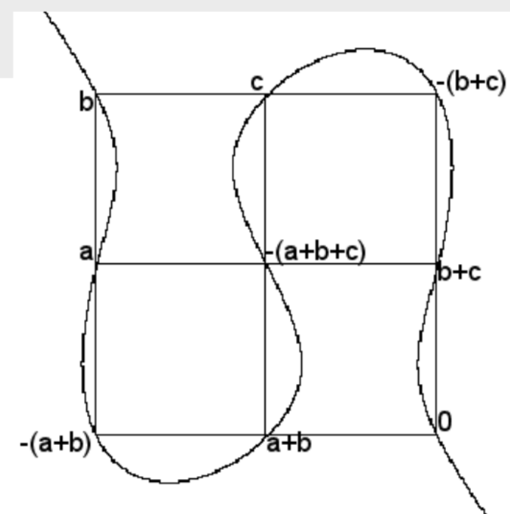
How Does ECC Work?





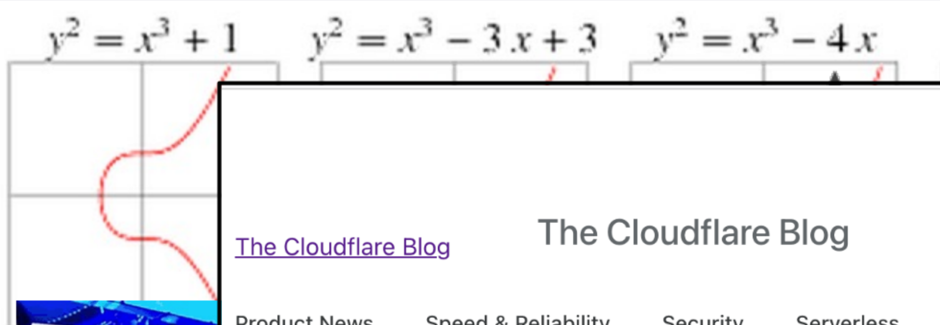


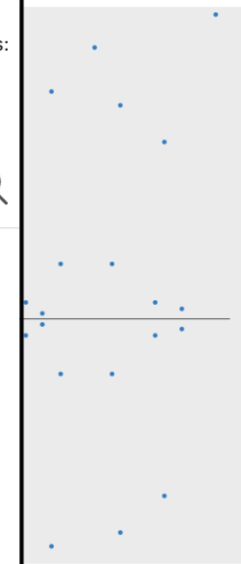


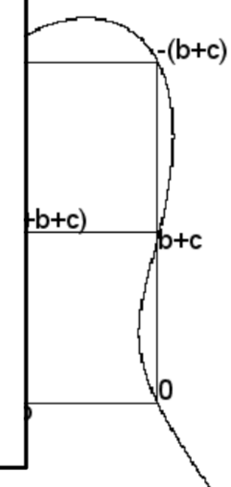


Me when people
try to explain
the ECC math

How Does ECC Work?







Contact Sales: +1 (888) 274-3482

Thanks for being here, come back soon. Get notified of new posts:

The Cloudflare Blog The Cloudflare Blog

Product News Speed & Reliability Security Serverless Zero Trust Developers Deep Dive Life @Cloudflare

A (Relatively Easy To Understand) Primer on Elliptic Curve Cryptography

10/23/2013

Nick Sullivan Nick Sullivan

Elliptic Curve Cryptography (ECC) is one of the most powerful but least understood types of cryptography in wide use today. At CloudFlare, we make extensive use of ECC to secure everything from our customers' HTTPS connections to how we pass data between our data centers.

Fundamentally, we believe it's important to be able to understand the technology behind any security system in order to trust it. To that end, we looked around to find a good, relatively easy-to-understand primer on ECC in order to share with our users.

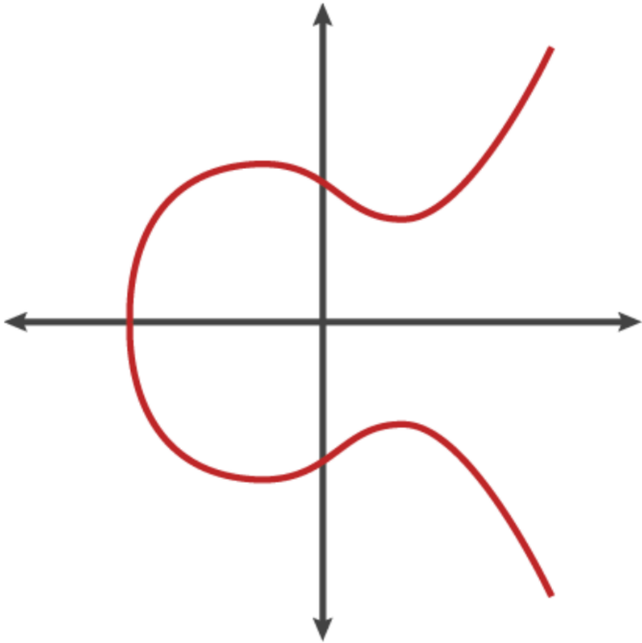
Me
tr
th

Elliptic Curve Cryptography

Elliptic Curve Cryptography (ECC) is a public-key scheme that provides the same operations via a different mechanism.

- Operates on elliptic curves over prime-order finite-fields
- Common curves are named
  - May/may not be descriptively named (i.e. “curveSM2”)

$y^2 = x^3 + ax + b$



Why Use ECC?

- Keys are significantly smaller
  - 256-bit vs. 3072-bit for 128-bit security
- Outputs are significantly smaller
- Attacks against ECC aren't as mature as those against RSA
- Significantly faster than finite-field

Table 3: OpenSSL 1.0.1c Speed Numbers with 64 bit ECC Optimizations

Certificate type	XLarge (c1.xlarge)				Medium (c1.medium)			
	Sign	Verify	Sign/s	Verify/s	Sign	Verify	Sign/s	Verify/s
RSA 2048 bits	0.002860s	0.000090s	349.7	11092.7	0.002925s	0.000092s	341.9	10863.7
256 bit ECDSA (nistp256)	0.0002s	0.0005s	4656.1	1848.7	0.0002s	0.0006s	4492.4	1773.6
384 bit ECDSA (nistp384)	0.0004s	0.0020s	2341.2	487.9	0.0004s	0.0021s	2269.4	470.2

Why Use ECC?

- Keys are significantly smaller
  - 256-bit vs. 3072-bit for 128-bit security
- Outputs are significantly smaller
- Attacks against ECC aren't **as mature** as those against RSA
- Significantly faster than finite-field

Table 3: OpenSSL 1.0.1c Speed Numbers with 64 bit ECC Optimizations

Certificate type	XLarge (c1.xlarge)				Medium (c1.medium)			
	Sign	Verify	Sign/s	Verify/s	Sign	Verify	Sign/s	Verify/s
RSA 2048 bits	0.002860s	0.000090s	349.7	11092.7	0.002925s	0.000092s	341.9	10863.7
256 bit ECDSA (nistp256)	0.0002s	0.0005s	4656.1	1848.7	0.0002s	0.0006s	4492.4	1773.6
384 bit ECDSA (nistp384)	0.0004s	0.0020s	2341.2	487.9	0.0004s	0.0021s	2269.4	470.2

Maybe-Safe ECC Curves

- CNSA approves use-specific curves

NSA | CSS National Security Agency | Central Security Service
Defending our Nation. Securing the Future.

Search

PROGRAMS › IAD INITIATIVES › COMMERCIAL NATIONAL SECURITY ALGORITHM SUITE

Commercial National Security Algorithm Suite

The Commercial National Security Algorithm Suite (CNSA Suite) will provide new algorithms for those customers who are looking for mitigations to perform, replacing the current Suite B algorithms.

In the current global environment, rapid and secure information sharing is important to protect our Nation, its citizens and its interests. Strong cryptographic algorithms and secure protocol standards are vital tools that contribute to our national security and help address the ubiquitous need for secure, interoperable communications.

View related Algorithm Guidance documents.

Currently, Suite B cryptographic algorithms are specified by

Transition Algorithms

Algorithm	Function	Specification	Parameters
Advanced Encryption Standard (AES)	Symmetric block cipher used for information protection	FIPS Pub 197	Use 256 bit keys to protect up to TOP SECRET
Elliptic Curve Diffie-Hellman (ECDH) Key Exchange	Asymmetric algorithm used for key establishment	NIST SP 800-56A	Use Curve P-384 to protect up to TOP SECRET.
Elliptic Curve Digital Signature Algorithm (ECDSA)	Asymmetric algorithm used for digital signatures	FIPS Pub 186-4	Use Curve P-384 to protect up to TOP SECRET.
Secure Hash Algorithm (SHA)	Algorithm used for computing a condensed representation of information	FIPS Pub 180-4	Use SHA-384 to protect up to TOP SECRET.

NIST Curves are Sketchy

Support the Guardian
Available for everyone, funded by readers
Contribute → Subscribe →

Sign in **The Guardian** For 200 years

News | Opinion | Sport | Culture | Lifestyle

US World Environment Soccer US Politics Business Tech Science Newsletters Green light

Glenn Greenwald on security and liberty The NSA files

This article is more than **7 years old**

Revealed: how US and UK spy agencies defeat internet privacy and security

- **NSA and GCHQ unlock encryption used to protect emails, banking and medical records**
- **$250m-a-year US program works covertly with tech companies to insert weaknesses into products**
- **Security experts say programs 'undermine the fabric of the internet'**

- **Q&A: submit your questions for our privacy experts**

▲ Through covert partnerships with tech companies, the spy agencies have inserted secret vulnerabilities into encryption software. Photograph: Kacper Pempel/Reuters

James Ball, Julian Borger *and* **Glenn Greenwald**

Fri 6 Sep 2013 06.24 EDT

Dual EC: A Standardized Back Door

Daniel J. Bernstein[1,2], Tanja Lange[1], and Ruben Niederhagen[1]

[1] Department of Mathematics and Computer Science
Technische Universiteit Eindhoven
P.O. Box 513, 5600 MB Eindhoven, The Netherlands
tanja@hyperelliptic.org, ruben@polycephaly.org

[2] Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607–7045, USA
djb@cr.yp.to

On the Practical Exploitability of Dual EC in TLS Implementations

Stephen Checkoway[1], Matthew Fredrikson[2], Ruben Niederhagen[3], Adam Everspaugh[2], Matthew Green[1], Tanja Lange[3], Thomas Ristenpart[2], Daniel J. Bernstein[3,4], Jake Maskiewicz[5], and Hovav Shacham[5]
[1] *Johns Hopkins University,* [2] *University of Wisconsin,* [3] *Technische Universiteit Eindhoven,* [4] *University of Illinois at Chicago,* [5] *UC San Diego*

Maybe-Safe ECC Curves

- CNSA approves use-specific curves

NSA | CSS National Security Agency | Central Security Service
Defending our Nation. Securing the Future.

Search

PROGRAMS › IAD INITIATIVES › COMMERCIAL NATIONAL SECURITY ALGORITHM SUITE

Com
Algo

The C
Suite)
are loc
Suite

In the
tion sh
and its
protoc
tional
cure, i

View related Algorithm Guidance documents.

Currently, Suite B cryptographic algorithms are specified by

Elliptic Curve Diffie-Hellman (ECDH) Key Exchange	Asymmetric algorithm used for key establishment	NIST SP 800-56A	Use Curve P-384 to protect up to TOP SECRET.
Elliptic Curve Digital Signature Algorithm (ECDSA)	Asymmetric algorithm used for digital signatures	FIPS Pub 186-4	Use Curve P-384 to protect up to TOP SECRET.

Likely-Safe ECC Curves

- Curve25519 (Ed25519, x25519, etc)
- Carefully chosen to remove common bugs/errors/vulns when using ECC

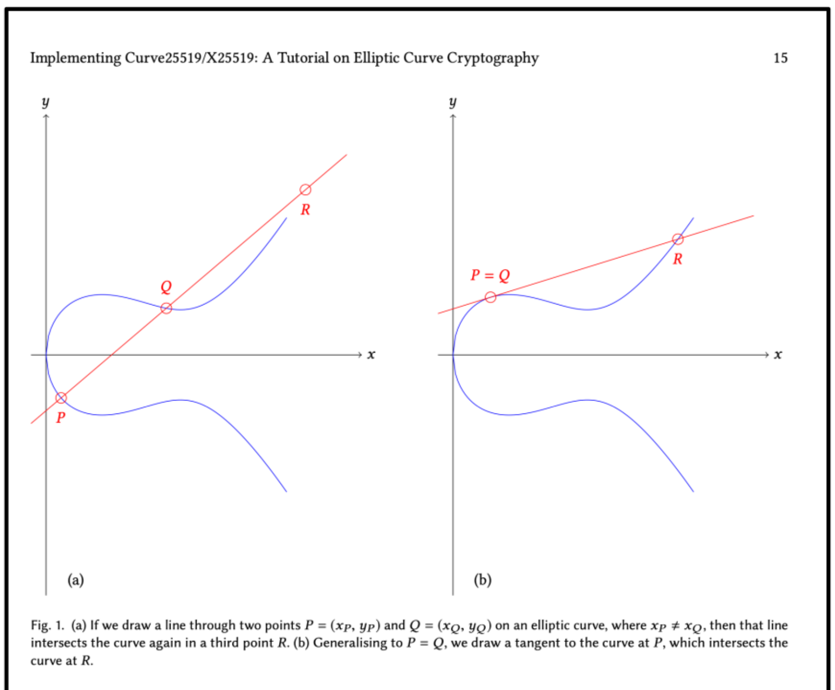
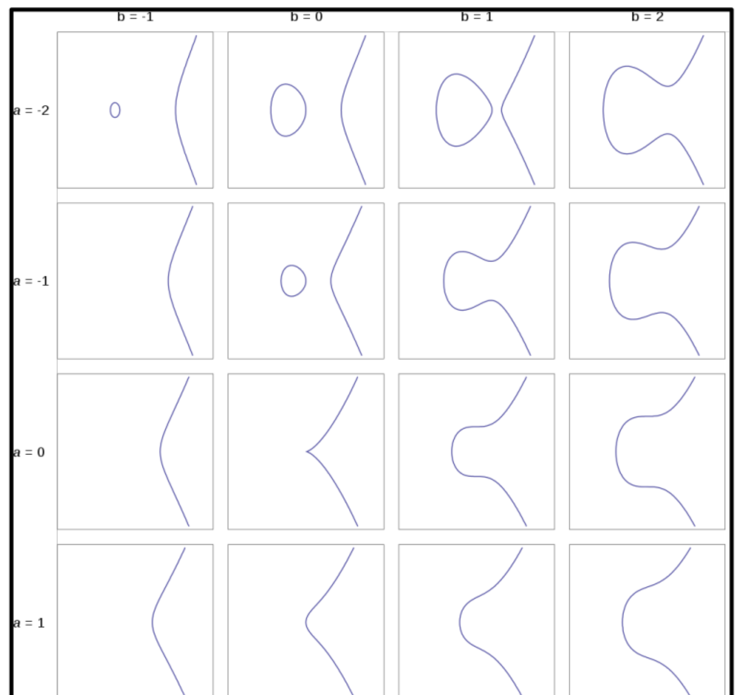
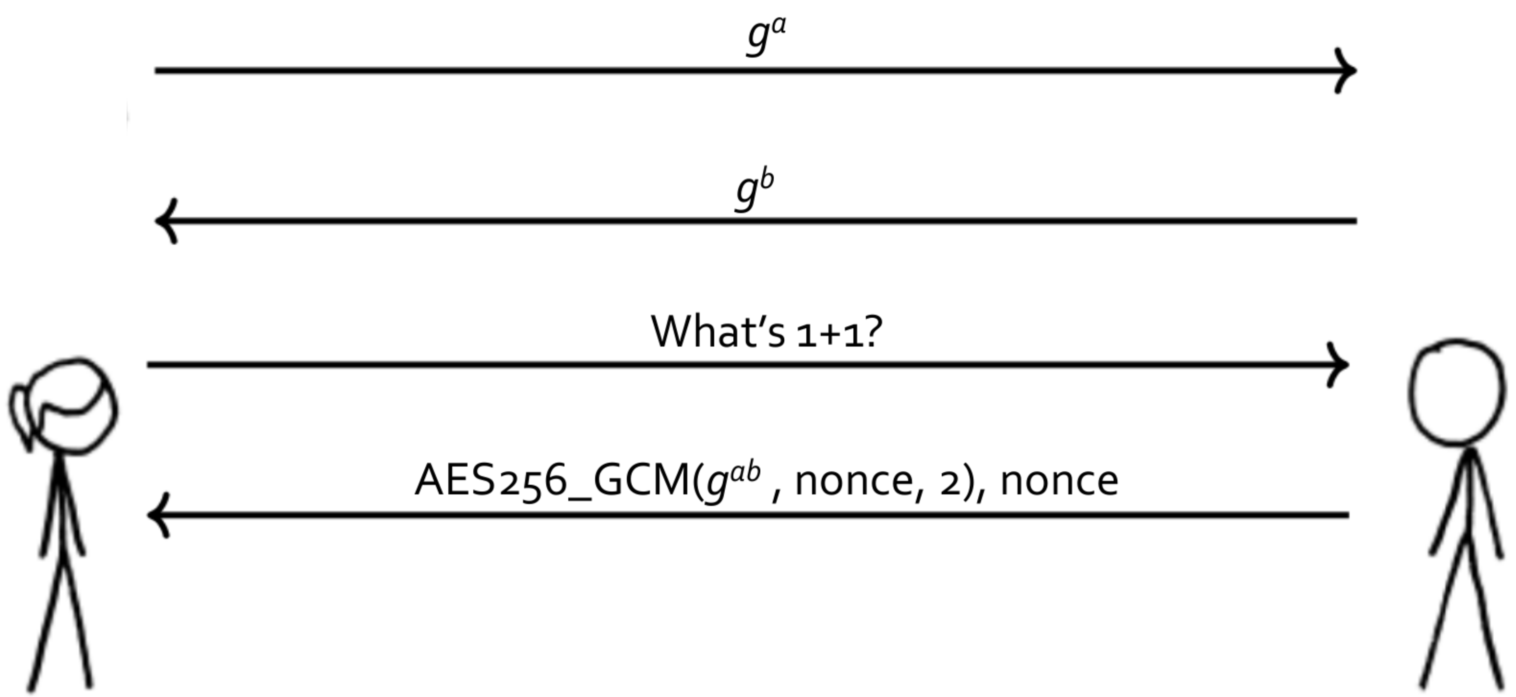

Implementing Curve25519/X25519: A Tutorial on Elliptic Curve Cryptography 15

Fig. 1. (a) If we draw a line through two points $P = (x_P, y_P)$ and $Q = (x_Q, y_Q)$ on an elliptic curve, where $x_P \neq x_Q$, then that line intersects the curve again in a third point R. (b) Generalising to $P = Q$, we draw a tangent to the curve at P, which intersects the curve at R.



Building a Secure Channel



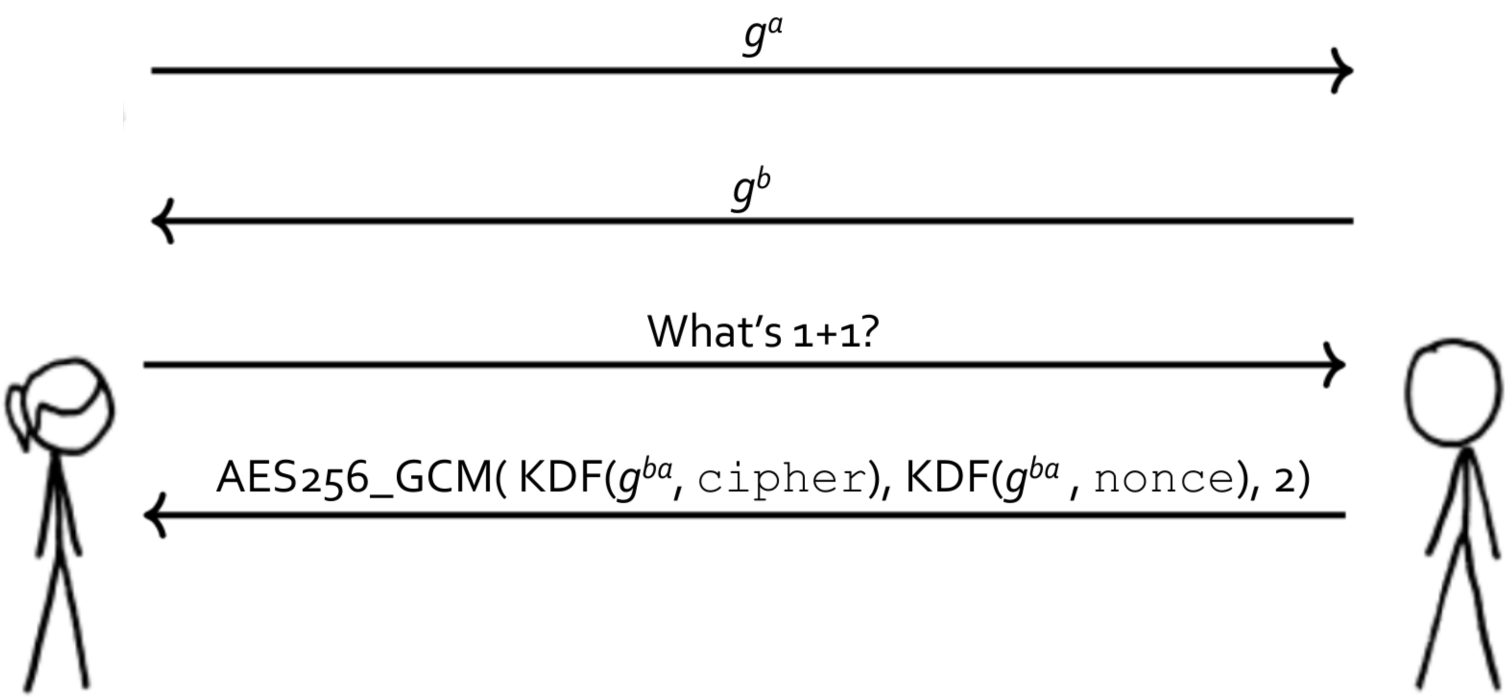
Key Derivation Function (KDF)

A **Key Derivation Function (KDF)** is one which can *safely* turn one shared-secret into multiple shared-secrets deterministically.

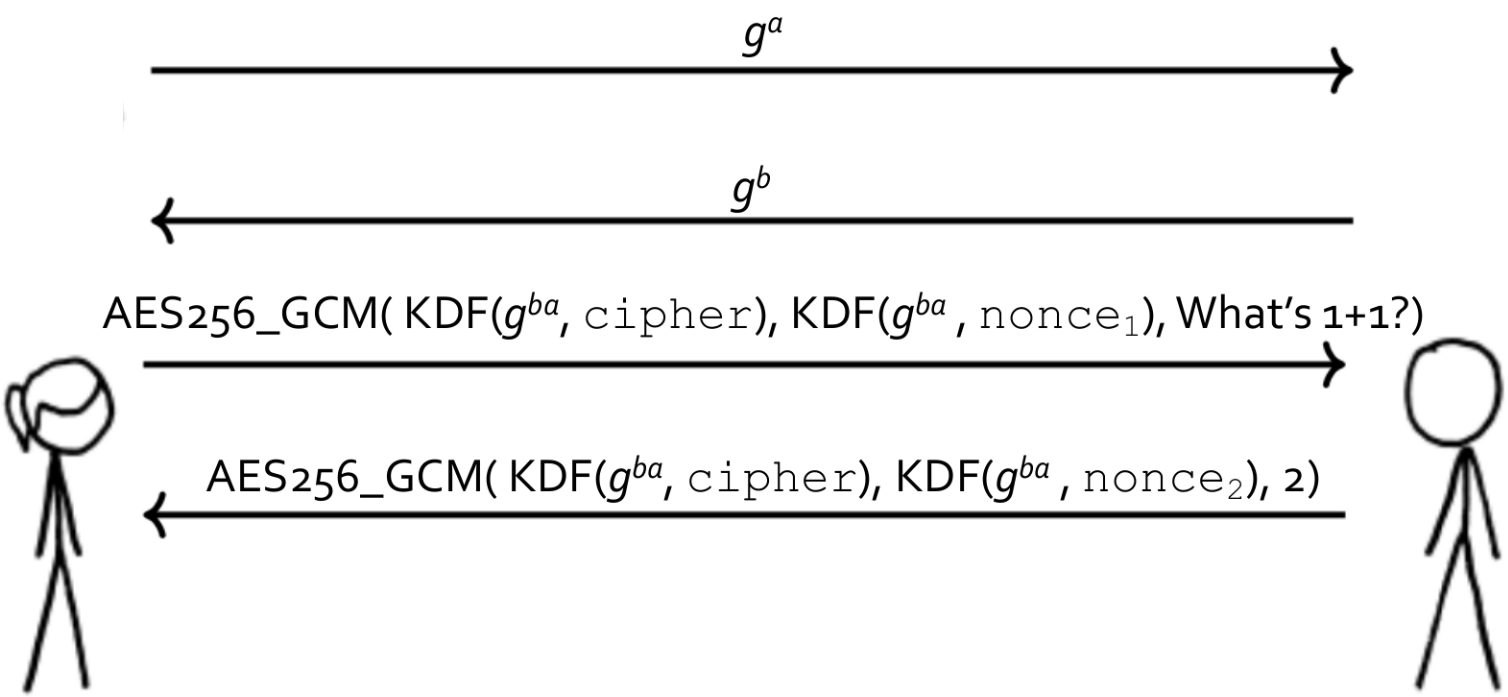
- HKDF is commonly used for protocols

```
PSK ->  HKDF-Extract = Early Secret
             |
             +-----> Derive-Secret(., "ext binder" | "res binder", "")
             |                     = binder_key
             |
             +-----> Derive-Secret(., "c e traffic", ClientHello)
             |                     = client_early_traffic_secret
             |
             +-----> Derive-Secret(., "e exp master", ClientHello)
             |                     = early_exporter_master_secret
             v
       Derive-Secret(., "derived", "")
```

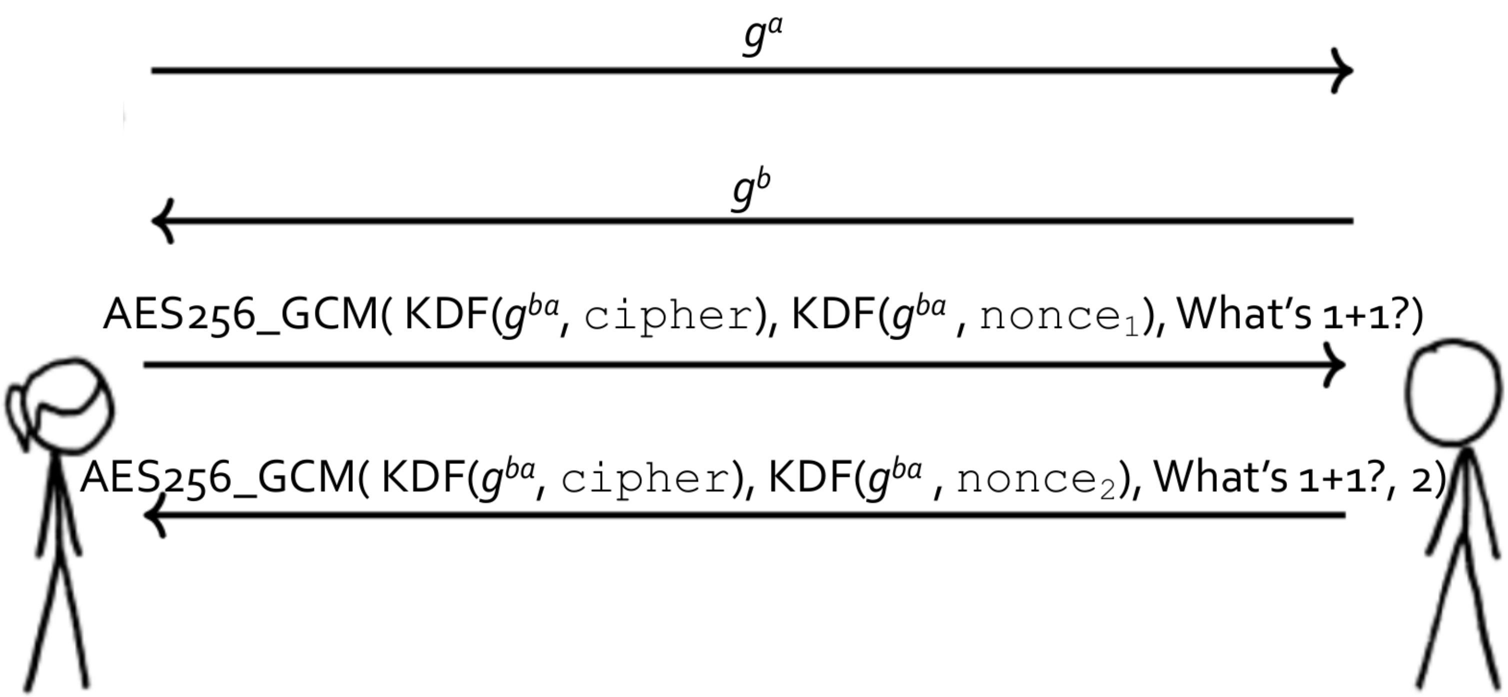
Building a Secure Channel



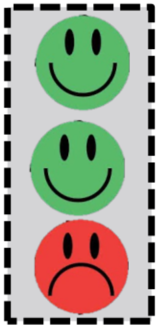
Building a Secure Channel



Building a Secure Channel



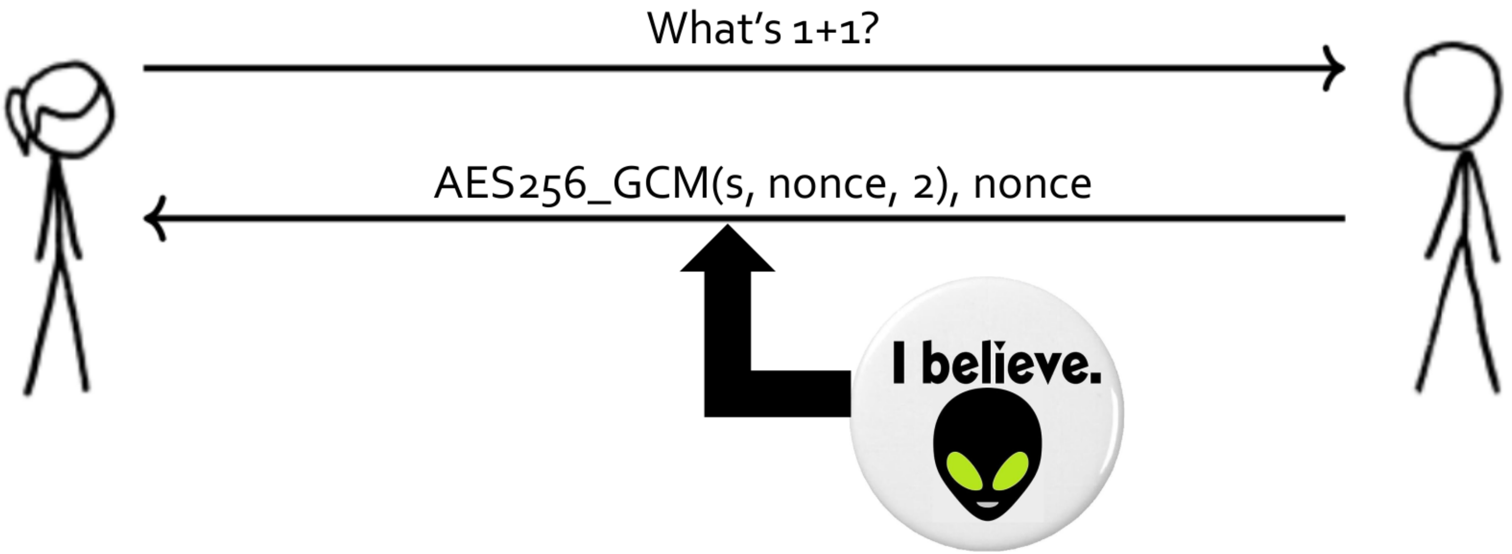
Building a Secure Channel



Confidentiality

Message Integrity

Sender Authenticity



Computer and Network Security

Lecture 06:
KEX & Asymmetric Operations

COMP-5370/6370
Fall 2024