CS 7480
Special Topics in PL

Class 2: Foundations

Sep 10, 2024

Joshua Gancher

Today

- Build a knowledge base to help you be successful in this class
- Foundations in Provable Cryptography

Cryptography

Scope for today:

Crypto used in communication protocols: TLS/WireGuard/Kerberos/…

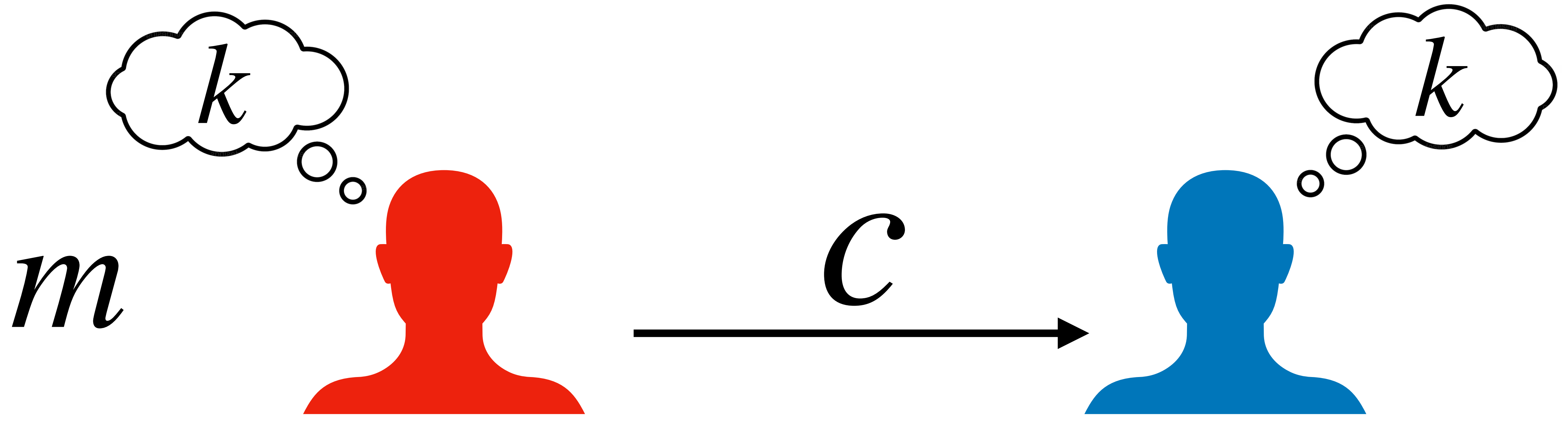
Everything you need to know to understand what we will be reading for Part 1

Won’t touch on “next-gen crypto”: Multi-Party Computation, Blockchains, ZKP, …

The Fundamental Problem

- Communication protocols are designed to work over **open networks**
  - Attacker can read and modify **any message**, control scheduling
- How to regain any security?
  - Parties must **hold secret keys**, take advantage of their secrecy

Key Primitive: Symmetric Encryption



$c \xleftarrow{\$} \mathsf{Enc}(k, m)$ $\quad\quad$ $m \leftarrow \mathsf{Dec}(k, c)$

Encryption Scheme:

Gen : probabilistically outputs a key $k \in \{0,1\}^*$

Enc : given key k and $m \in \{0,1\}^*$,
probabilistically outputs **ciphertext** $c \in \{0,1\}^*$

Dec : given key k and ciphertext c
outputs message **or returns an error** $\perp$

Security Properties for Symmetric Encryption
(informally, for Authenticated Encryption)

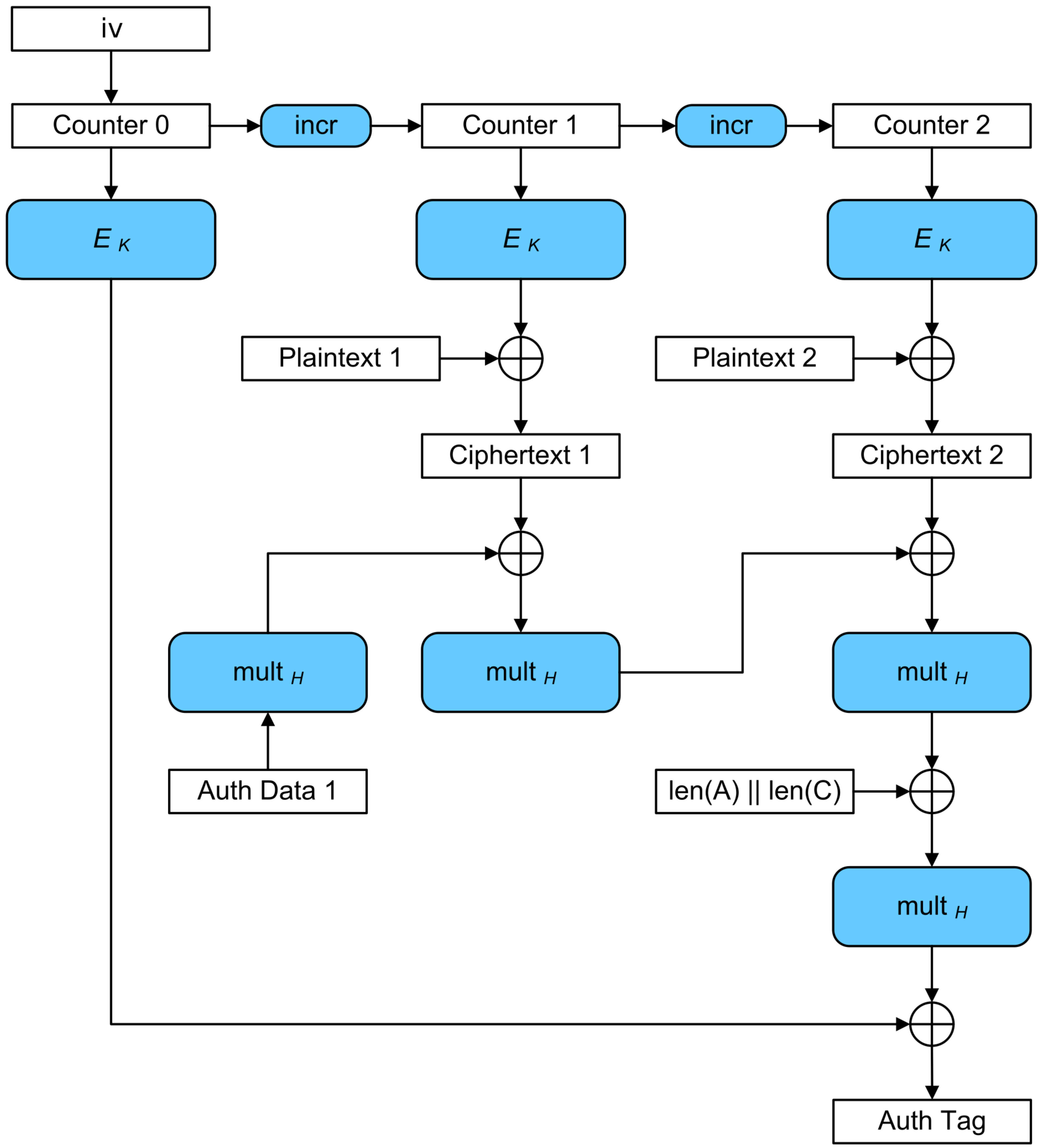
Decryption must be **correct**:

if k <- Gen and c <- Enc(k, m), then Dec(k, c) = m

If attacker only sees ciphertexts but not otherwise the key:

- Plaintext stays secret, except for its length (Semantic Security)
- Attacker can’t produce a valid ciphertext (Ciphertext Integrity)

Representative Instance: AES-GCM



Source: Wikipedia

Symmetric Crypto:

built on decades of research

into secure ciphers, hashes

security is heuristic (but heavily studied)

Keys are uniformly random bytes

Main Problem: Key Distribution

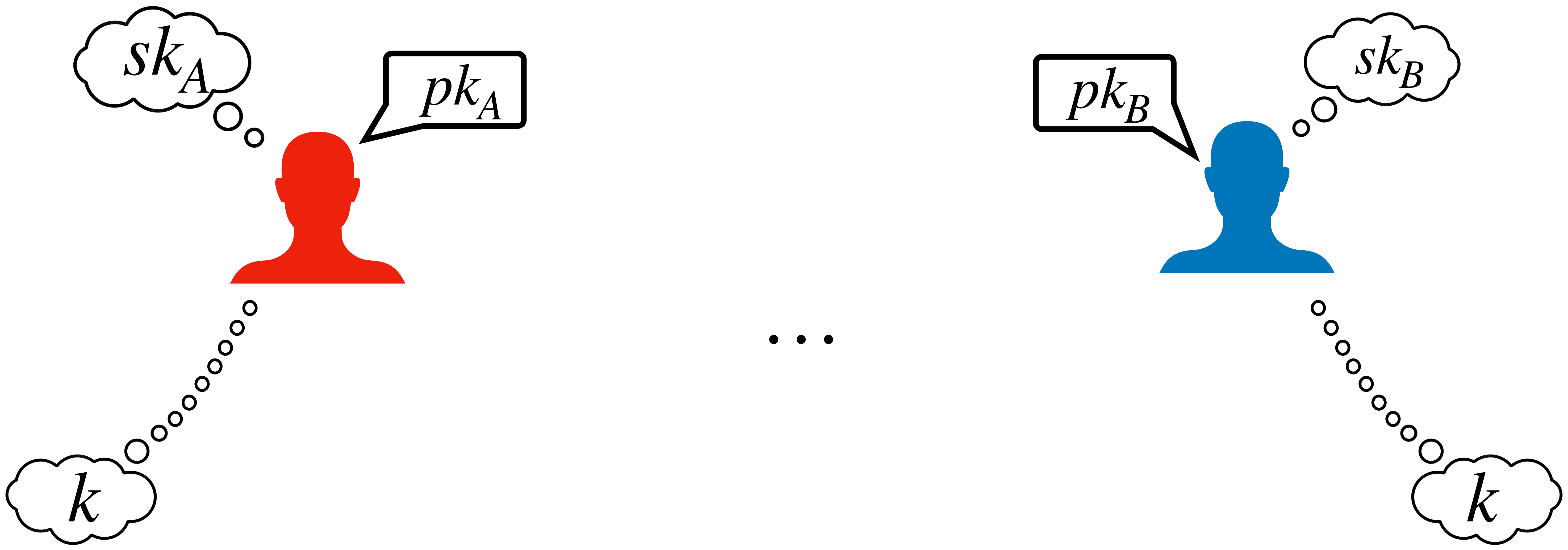
Symmetric encryption only secure if

both parties hold the same secret, randomly generated key

- If key isn't a secret: attacker can decrypt, forge ciphertexts
- If key isn't the same: communication will fail, or parties can be deceived
  - A thinks it's talking to B, but really talking to C
- If key isn't generated correctly, attacker can use this to attack
  - If first bit = 0, now there are **half** as many possible keys!

Solutions to Key Generation

1. Manually deliver the key to both people
    - Unsuitable for the public internet
2. Use more cryptography to **deliver** the key securely
    - We do this by using **public key cryptography** (PKC)



Crypto Necessary for Communication Protocols

Encryption

carries out secure comm.

Key Derivation

converts shared secrets to good encryption keys

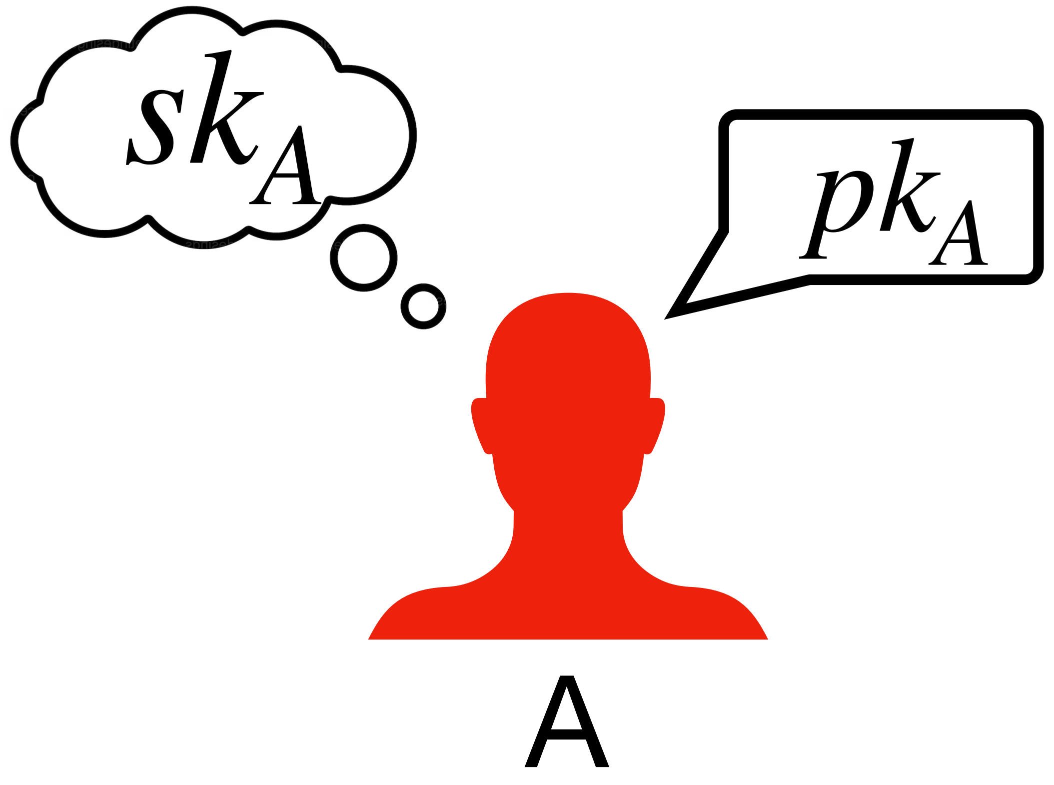
Diffie-Hellman

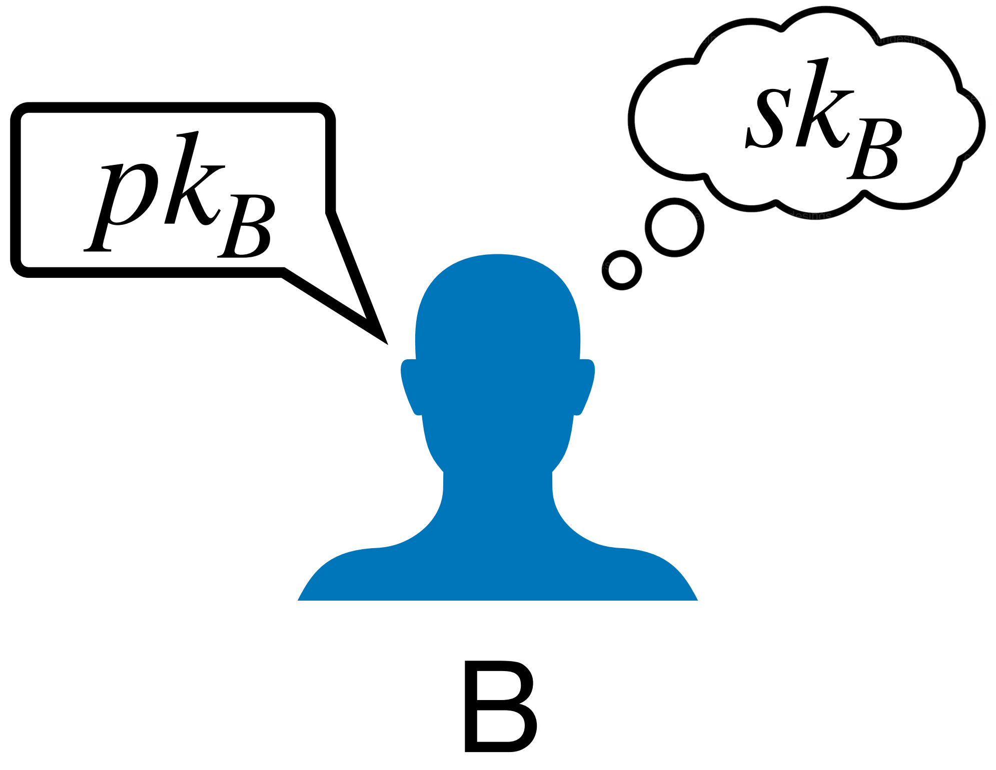
create **shared secret** using PKC

Digital Signatures

authenticating against public key

Diffie-Hellman Exchange





Only A knows A’s secret key

Only B knows B’s secret key

Over public channel,

construct **shared secret** that only A and B know

Diffie-Hellman Exchange

$$\text{Gen}() : \text{SecretKey}$$

$$\text{MakePK}(sk : \text{SecretKey}) : \text{PublicKey}$$

$$\text{combine}(sk : \text{SecretKey}, pk : \text{PublicKey}) : \text{SharedSecret}$$

my secret key ↗ (sk)

their public key ↖ (pk)

$$ss_A := \text{combine}(sk_A, pk_B) \qquad ss_B := \text{combine}(sk_B, pk_A)$$

Diffie-Hellman Exchange Properties

correctness: $\text{combine}(sk_A, pk_B) = \text{combine}(sk_B, pk_A)$

secrecy: if sk_A, sk_B are kept secret,
ss looks random to attacker

$ss_A := \text{combine}(sk_A, pk_B)$

$ss_B := \text{combine}(sk_B, pk_A)$

Diffie-Hellman

under the hood: cyclic groups (often Elliptic Curves)

G cyclic group of order N, generator g

$$\text{Gen}() := \text{Uniform}(\mathbb{Z}_N)$$
$$\text{MakePK}(x) := g^x$$

$$\text{combine}(x, h) := h^x$$

$$\text{combine}(sk_B, pk_A) = (g^b)^a = g^{ba} = (g^a)^b = \text{combine}(sk_A, pk_B)$$

Diffie-Hellman Assumption: g^{ab} looks random given g^a, g^b

Problem #1: Using the Shared Secret

$ss = \text{combine}(sk_B, pk_A)$ has **high entropy** (unguessable), low **uniformity**

Unsuitable for encryption keys!

Examples: K' = 010101 || K, K uniformly random

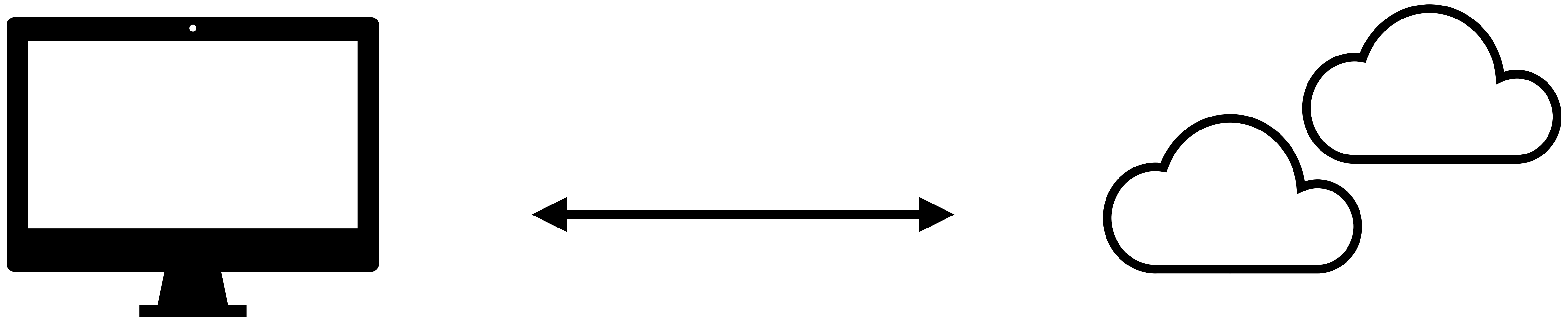
K' = K[0] || 0 || K[1] || 0 || … || K[N]

g^{xy}, where x, y uniform

Solution: **Key Derivation Function**

unguessable but non-uniform → KDF → uniformly random bytes

Problem #2: Getting the right public key

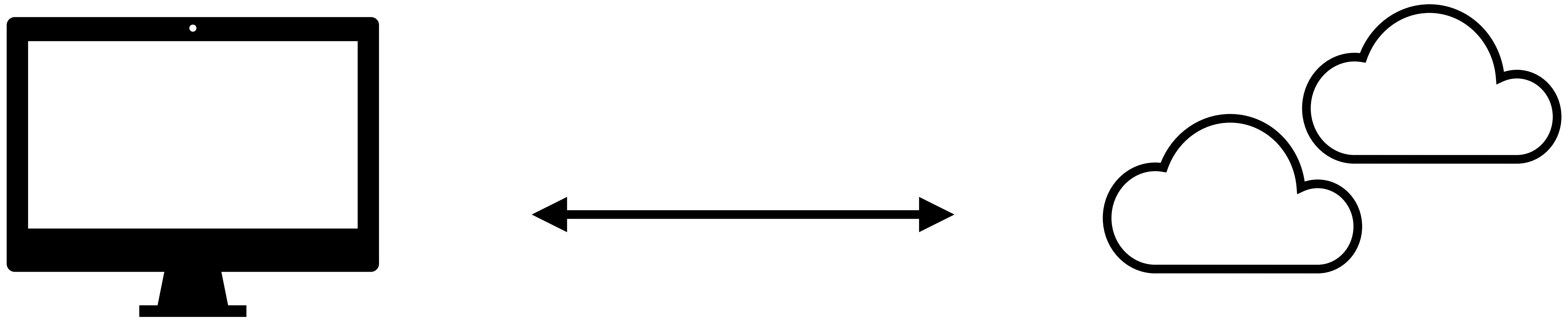


How do I discover Server's public key?

Similar to problem for encryption keys, but big difference:

Distributing **public information** rather than **secret**

Problem #2: Getting the right public key



Option 1: Manually plug in the public key

WIREGUARD
FAST, MODERN, SECURE VPN TUNNEL

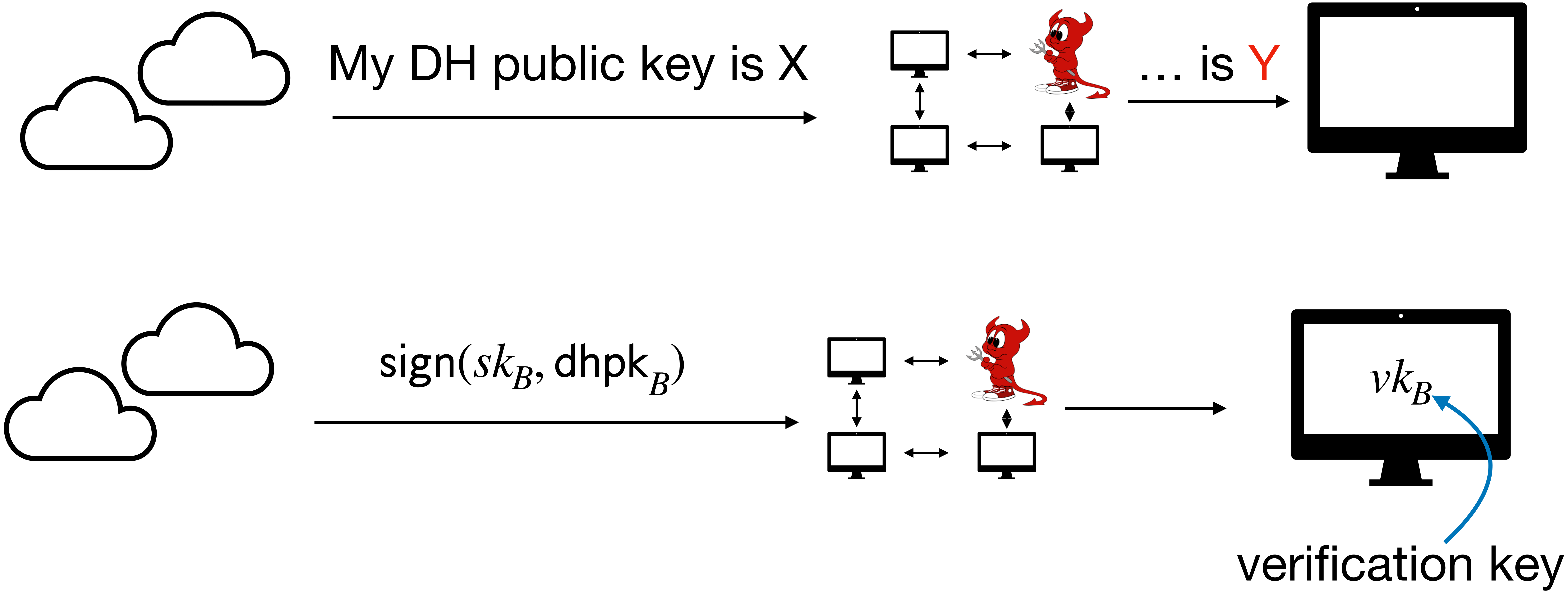
Option 2: **Digitally sign** the public key with another one

TLS

Allows DH key to change, but signing key stays the same

Digital Signatures

- Proves **authenticity** of data against public **verification key**



Digital Signatures

$$\text{Gen}() : \text{SigningKey}$$

$$\text{MakeVK}(sk : \text{SigningKey}) : \text{VerifKey}$$

$$\text{Sign}(sk : \text{SigningKey}, m \in \{0,1\}^*) : \text{Signature}$$

$$\text{Verify}(vk : \text{VerifKey}, m \in \{0,1\}^*, s : \text{Signature}) : \text{Bool}$$

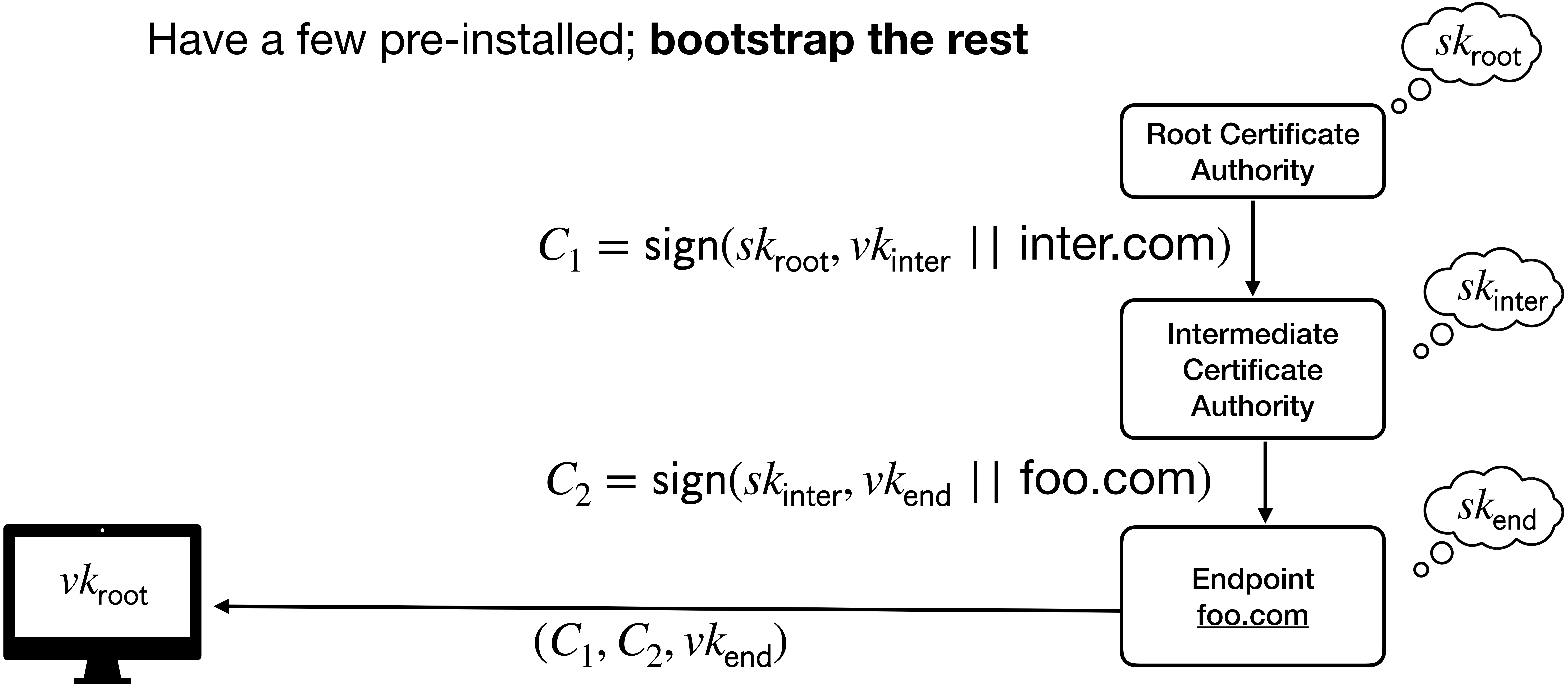
unforgeability: attacker cannot forge signatures

using only public verification key, prior signatures

Getting Correct Verification Keys

How to install verification key?

Have a few pre-installed; **bootstrap the rest**



rary/Security/Certificates.bundle/Contents/Resources/TrustStore.html

Trust store version: 2024040500

Trusted Certificates

Certificate name	Issued by	Type	Key size	Sig alg	Serial number	Expires	EV policy	Fingerprint (SHA-256)
AAA Certificate Services	AAA Certificate Services	RSA	2048 bits	SHA-1	01	23:59:59 Dec 31, 2028	Not EV	D7 A7 A0 FB 5D 7E 27 31 D7 71 E9 48 4E BC DE F7 1D 5F 0C 3E 0A 29 48 78 2B C8 3E E0 EA 69 9E F4
AC RAIZ FNMT-RCM	AC RAIZ FNMT-RCM	RSA	4096 bits	SHA-256	5D 93 8D 30 67 36 C8 06 1D 1A C7 54 84 69 07	00:00:00 Jan 1, 2030	Not EV	EB C5 57 0C 29 01 8C 4D 67 B1 AA 12 7B AF 12 F7 03 B4 61 1E BC 17 B7 DA B5 57 38 94 17 9B 93 FA
USERTrust RSA Certification Authority	USERTrust RSA Certification Authority	RSA	4096 bits	SHA-384	01 FD 6D 30 FC A3 CA 51 A8 1B BC 64 0E 35 03 2D	23:59:59 Jan 18, 2038	1.3.6.1.4.1.6449.1.2.1.5.1 2.23.140.1.1	E7 93 C9 B0 2F D8 AA 13 E2 1C 31 22 8A CC B0 81 19 64 3B 74 9C 89 89 64 B1 74 6D 46 C3 D4 CB D2

USERTrust RSA Certification Authority
- InCommon RSA Server CA
  - khoury.northeastern.edu

khoury.northeastern.edu
Issued by: InCommon RSA Server CA
Expires: Saturday, October 5, 2024 at 7:59:59 PM Eastern Daylight Time
This certificate is valid

Trust

Details

Subject Name

Country or Region	US
State/Province	Massachusetts
Organization	Northeastern University
Common Name	khoury.northeastern.edu

Issuer Name

Country or Region	US
State/Province	MI
Locality	Ann Arbor
Organization	Internet2
Organizational Unit	InCommon
Common Name	InCommon RSA Server CA

Crypto Necessary for Communication Protocols

Encryption
carries out secure comm.

Key Derivation
converts shared secrets to good encryption keys

Diffie-Hellman
create **shared secret** using PKC

Digital Signatures
authenticating against public key

Specifying and Proving Security

- Up until now: a tour of cryptographic algorithms
- Rest of today: how do we make sure they're used correctly?

… → Formal specs of security for encryption, signatures, … → Formal specs of security for protocols

Two Camps for Protocol Security

Symbolic Crypto

or, “Dolev-Yao” model

Represent crypto using abstract terms, equations

Closer to existing Formal Methods tools (e.g., Model Checking)

Weaker security guarantees

Computational Crypto

Same framework as modern cryptography: complexity theory

Requires more custom tool support

Accurate security guarantees

Symbolic Crypto

Protocol messages come from BNF grammar

$$t ::= K \in \mathsf{Keys} \mid (t, t) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid \mathsf{enc}(t, t) \mid \mathsf{dec}(t, t) \mid C \in \mathsf{Const} \mid \ldots$$

Example term:

$$\mathsf{fst}((\mathsf{enc}(K_1, K_2), \mathsf{dec}(K_3, 42)))$$

Symbolic Crypto

Protocol messages come from BNF grammar

$$t ::= K \in \mathsf{Keys} \mid (t, t) \mid \mathsf{fst}(t) \mid \mathsf{snd}(t) \mid \mathsf{enc}(t, t) \mid \mathsf{dec}(t, t) \mid C \in \mathsf{Const} \mid \ldots$$

Subject to **equations**:

$$\mathsf{dec}(k, \mathsf{enc}(k, m)) = m$$

(reversed argument order from Tamarin)

$$\mathsf{fst}((x, y)) = x$$

$$\mathsf{snd}((x, y)) = y$$

Attacker allowed to build

arbitrary terms using its current knowledge

Attacker's Knowledge Set

$A(t)$: attacker can deduce t

$$\frac{A(t_1) \quad A(t_2)}{A(\mathsf{enc}(t_1, t_2))} \qquad \frac{A(t_1) \quad A(t_2)}{A(\mathsf{dec}(t_1, t_2))} \qquad \frac{A(t)}{A(\mathsf{fst}(t))} \qquad \frac{A(t)}{A(\mathsf{snd}(t))} \qquad \frac{}{A(C)}$$

If S is a set of terms, let $\text{Clo}(S)$ be the **closure** of S

$\text{Clo}(\{K_1, K_2\})$ = all terms built out of K_1, K_2

Example: is $K_1 \in \text{Clo}(\{K_2, \text{enc}(K_2, K_1)\})$?

Attacker's Knowledge Set

$A(t)$: attacker can deduce t

$$\frac{A(t_1) \quad A(t_2)}{A(\mathsf{enc}(t_1, t_2))} \qquad \frac{A(t_1) \quad A(t_2)}{A(\mathsf{dec}(t_1, t_2))} \qquad \frac{A(t)}{A(\mathsf{fst}(t))} \qquad \frac{A(t)}{A(\mathsf{snd}(t))} \qquad \frac{}{A(C)}$$

Let $S = \{\mathsf{enc}(K_3, K_2), \mathsf{enc}(K_2, K_1), \mathsf{enc}(K_2, m), K_3\}$

Is $\mathsf{enc}(K_1, m) \in \mathsf{Clo}(S)$?

Let $S = \{\mathsf{enc}(K_3, K_2), \mathsf{enc}(K_2, K_1), \mathsf{enc}(K_2, m), K_3\}$

Is $\mathsf{enc}(K_3, m) \in \mathsf{Clo}(S)$?

Example Protocol

Alice =

```
output enc(K, N);
recv c;
if let m = dec(K, c) then
  if m == N + 1 then
  output “ok”
else output “fail”
```

Bob =

```
recv c;
if let x = dec(K, c) then
  output enc(K, x + 1)
else skip
```

Attacker may query parties arbitrarily, subject to its knowledge

Trace event (e.g., input/output)

Generates a **trace** $(A_0, P_0) \xrightarrow{e_0} \ldots \xrightarrow{e_n} (A_n, P_n)$

Knowledge of attacker

Current state of protocol participant(s)

Example Protocol

Alice

```
output enc(K, N);
recv c;
if let m = dec(K, c) then
  if m == N + 1 then
  output “ok”
else output “fail”
```

Bob

```
recv c;
if let x = dec(K, c) then
  output enc(K, x + 1)
else skip
```

enc(K, N) →

Attacker Knowledge

$\{\}$

Example Protocol

Alice

```
recv c;
if let m = dec(K, c) then
  if m == N + 1 then
  output “ok”
else output “fail”
```

Attacker Knowledge

$$\{\text{enc}(K, N)\}$$

`deliver enc(K, N) to Bob` ←

Bob

```
recv c;
if let x = dec(K, c) then
  output enc(K, x + 1)
else skip
```

Example Protocol

Alice

```
recv c;
if let m = dec(K, c) then
  if m == N + 1 then
  output “ok”
else output “fail”
```

Bob

```
if let x = dec(K, enc(K, N)) then
  output enc(K, x + 1)
else skip
```

Attacker Knowledge

$$\{\mathsf{enc}(K, N)\}$$

Example Protocol

Alice

```
recv c;
if let m = dec(K, c) then
  if m == N + 1 then
  output “ok”
else output “fail”
```

enc(K, N + 1)

→

Attacker Knowledge

$\{\text{enc}(K, N)\}$

Bob

```
output enc(K, N + 1)
```

Example Protocol

Alice

```
recv c;
if let m = dec(K, c) then
  if m == N + 1 then
  output “ok”
else output “fail”
```

Attacker Knowledge

$$\{\mathsf{enc}(K, N), \mathsf{enc}(K, N+1)\}$$

```
deliver enc(K, N + 1) to Alice
```

←

Bob

```
skip
```

Example Protocol

Alice

```
if let m = dec(K, enc(K, N + 1)) then
  if m == N + 1 then
  output “ok”
else output “fail”
```

Bob

```
skip
```

Attacker Knowledge

$$\{\mathsf{enc}(K, N), \mathsf{enc}(K, N + 1)\}$$

Example Protocol

Alice

```
output “ok”
```

Bob

```
skip
```

Attacker Knowledge

$$\{\mathsf{enc}(K, N), \mathsf{enc}(K, N+1)\}$$

Example Protocol: Properties

Secrecy: if initial attacker knowledge $A_0 = \{\}$, can attacker learn K?

$$(A_0, P_0) \rightarrow^* (A_n, P_n) \implies K \notin \text{Clo}(A_n)$$

Authentication:

If $A_0 = \{\}$, and Alice output “ok”, Bob must have output enc(K, N + 1)

$$(A_0, P_0) \xrightarrow{T} (A_n, \text{done}) \wedge (\text{Alice : ok}) \in T \implies (\text{Bob : enc}(K, N + 1)) \in T$$

How to Prove Symbolic Properties

Tamarin Prover

The Tamarin prover is a security protocol verification tool that supports both falsification (attack finding) and unbounded verification (proving) in the symbolic model. Security protocols are specified as multiset rewriting systems and analyzed with respect to temporal first-order properties.

Tamarin has been successfully used to analyze and support the development of modern security protocols [1,2], including TLS 1.3 [3,4], 5G-AKA [5,6], Noise [7], EMV (Chip-and-pin) [8], and Apple iMessage [9].

Get the docs | Install Tamarin

Weaknesses of Symbolic Crypto

- Encryption is assumed to be **completely opaque**:
  - The **only** thing attacker can do is manipulate abstract term
  - In the real world, attacker can:
    - View/change individual bits
    - Leverage brute-force attacks
    - View lengths of all messages
    - …

Weaknesses of Symbolic Crypto

NEWS

'CRIME' attack abuses SSL/TLS data compression feature to hijack HTTPS sessions

SSL/TLS data compression leaks information that can be used to decrypt HTTPS session cookies, researchers say

```
C = encrypt(compress(M))
```

length of C

⟶ **(partial) value of M**

Computational Model

- Same model as is assumed in crypto papers
- Cryptography modeled as **probabilistic algorithms on bitstrings**
- Attacker is **arbitrary probabilistic algorithm**

- To rule out brute-force attacks:
  - Attacker is **polynomial time**
  - Security guarantees must hold **up to negligible error**

Negligible Functions

Let A be a probabilistic poly-time (PPT) algorithm

Suppose A is trying to forge a ciphertext c so that $\mathsf{dec}(K, c)$ succeeds **without** knowing K or any valid ciphertext for K

Trivial attack:

1. Guess K by flipping random coins
2. Encrypt 0 using K

$$\Pr[A \ \mathsf{wins}] = \frac{1}{2^\lambda}$$

$\lambda = \mathsf{security\ parameter}$
(here, length of key)

Negligible Functions

Allow adversaries to violate security with **probability negligible in** λ

$\epsilon(\cdot)$ is negligible when for **all** polynomials P, there exists an N,

$$\lambda > N \implies \epsilon(\lambda) < \frac{1}{P(\lambda)}$$

Example: $\frac{1}{2^\lambda}$ is negligible

$\frac{1}{\lambda^{100}}$ is not

$\frac{\lambda^5}{2^\lambda}$ is negligible

Modelling Encryption Computationally

The Joy of Cryptography

by Mike Rosulek • joyofcryptography.com •

The Joy of Cryptography is a **free** undergraduate-level textbook that introduces students to the fundamentals of provable security.

Get the full PDF (4.1MB)

Latest draft: Jan 3, 2021; 286 pages

Security Properties for Symmetric Encryption
(informally, for Authenticated Encryption)

Decryption must be **correct**:

if k <- Gen and c <- Enc(k, m), then Dec(k, c) = m

If attacker only sees ciphertexts but not otherwise the key:

- Plaintext stays secret, except for its length (Semantic Security)
- Attacker can't produce a valid ciphertext (Ciphertext Integrity)

Security Games

Security specified via **indistinguishability** of programs (**security games**)

EncReal

init : $K \xleftarrow{\$} \text{Gen}(\lambda)$

Enc(m) :

return enc(K, m)

Dec(c) :

return dec(K, c)

initialization code

oracles:

input, return bitstrings

everything implicitly parameterized by λ

Adversaries for Games

EncReal

init : $K \xleftarrow{\$} \text{Gen}(\lambda)$

Enc(m) :

return enc(K, m)

Dec(c) :

return dec(K, c)

probabilistic algorithm A is an **adversary** for EncGame if:

- it is polytime in λ
- it makes queries to Enc, Dec
- eventually, returns a boolean b

$A \bowtie \text{EncGame}$: final **decision bit** output by A, when linked with EncGame

Indistinguishability of Games

EncReal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$

$\mathsf{Enc}(m)$:

return $\mathsf{enc}(K, m)$

$\mathsf{Dec}(c)$:

return $\mathsf{dec}(K, c)$

$$\approx$$

EncIdeal

$\cdots$

$\mathsf{Enc}(m)$:

$\cdots$

$\mathsf{Dec}(c)$:

$\cdots$

Security Game Indistinguishability:

No adversary can tell the difference between the two games

Indistinguishability of Games

EncReal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$

$\mathsf{Enc}(m)$:

return $\mathsf{enc}(K, m)$

$\mathsf{Dec}(c)$:

return $\mathsf{dec}(K, c)$

$\approx$

EncIdeal

$\cdots$

$\mathsf{Enc}(m)$:

$\cdots$

$\mathsf{Dec}(c)$:

$\cdots$

Security Game Indistinguishability:

$$\forall \text{ PPT } A, |\Pr[A \bowtie \mathsf{EncGame} = 1] - \Pr[A \bowtie \mathsf{EncGame}' = 1]| \leq \epsilon(\lambda)$$

where ϵ is negligible

Authenticated Encryption, Computationally

EncReal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$

Enc(m) :

return enc(K, m)

Dec(c) :

return dec(K, c)

$\approx$

EncIdeal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$
log := []

Enc(m) :

$c \leftarrow$ enc(K,$0^{|m|}$)
log[c] := m
return c

Dec(c) :

if log[c] exists :
  return log[c]
else return $\perp$

Authenticated Encryption, Computationally

Main idea: attacker only sees junk ciphertexts

Encryption cannot hide lengths, so

junk ciphertext must not obscure length

use an **ideal log** to map junk ciphertexts

back to real messages

EncIdeal

```
init :  K <-$ Gen(λ)
        log := []

Enc(m) :
  c ← enc(K, 0^|m|)
  log[c] := m
  return c

Dec(c) :
  if log[c] exists :
    return log[c]
  else return ⊥
```

Authenticated Encryption, Computationally

correctness: decryption is correct by construction

secrecy: junk ciphertexts don't hold message

integrity: attacker cannot create a ciphertext

except by calling Enc oracle; forgery impossible

EncIdeal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$
$\mathsf{log} := []$

Enc(m) :

$c \leftarrow \mathsf{enc}(K, 0^{|m|})$
$\mathsf{log}[c] := m$
return c

Dec(c) :

if $\mathsf{log}[c]$ exists :
return $\mathsf{log}[c]$
else return $\perp$

Authenticated Encryption, Computationally

Gen, enc, dec

is a **secure**

authenticated encryption scheme

if the two games are

indistinguishable

EncReal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$

Enc(m) :

return enc(K, m)

Dec(c) :

return dec(K, c)

$\approx$

EncIdeal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$
log := []

Enc(m) :

$c \leftarrow$ enc($K, 0^{|m|}$)
log[c] := m
return c

Dec(c) :

if log[c] exists :
return log[c]
else return $\perp$

Authenticated Encryption, Computationally

Often phrased as two separate equivalences:

- INT-CTXT (Ciphertext Integrity)
  - Ciphertexts are unforgeable
- IND-CPA (Semantic Security)
  - Ciphertexts keep messages secret

EncReal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$

Enc(m) :

return enc(K, m)

Dec(c) :

return dec(K, c)

$\approx$

EncIdeal

init : $K \xleftarrow{\$} \mathsf{Gen}(\lambda)$
log := []

Enc(m) :

$c \leftarrow$ enc($K, 0^{|m|}$)
log[c] := m
return c

Dec(c) :

if log[c] exists :
return log[c]
else return $\bot$

Using Authenticated Encryption

$$P_{\text{Real}}$$

Alice =

```
output enc(K, N);
```

Bob =

```
recv c;
if let x = dec(K, c) then
  output enc(K, x + 1)
else skip
```

Goal: attacker can't learn **anything** about N

Using Authenticated Encryption

sim has **no access** to

secrets at all!

simulator

$$\text{Goal}: \quad \forall A, \exists S, A \bowtie P_{\text{Real}} = S$$

adversary

adv interacting

with secrets

Simulation:

“any information that can be computed using protocol

can be computed without the protocol”

Using Authenticated Encryption

$$P_{\text{Real}}$$

Alice =
```
output enc(K, N);
```

Bob =
```
recv c;
if let x = dec(K, c) then
  output enc(K, x + 1)
else skip
```

Idea: **rewrite** protocol to make use of security game for enc, dec

Using Authenticated Encryption

$$P_{\text{Hybrid}}$$

Alice =
```
output G.Enc(N);
```

Bob =
```
recv c;
if let x = G.Dec(c) then
  output G.Enc(x + 1)
else skip
```

Idea: **rewrite** protocol to make use of security game for enc, dec

Using Authenticated Encryption

$P_{\mathsf{Real}} \approx P_{\mathsf{Hybrid}} \bowtie \mathtt{EncReal}$ by unfolding definitions

$\mathtt{EncReal} \approx \mathtt{EncIdeal}$ by assumption

$P_{\mathsf{Real}} \approx P_{\mathsf{Hybrid}} \bowtie \mathtt{EncIdeal}$ by **congruence**

Hybrid Protocol

$$P_{\text{Hybrid}}$$

Alice =
```
output G.Enc(N);
```

Bob =
```
recv c;
if let x = G.Dec(c) then
  output G.Enc(x + 1)
else skip
```

Idealized Protocol

no dataflow from N to output!

$$P_{\text{Ideal}} := P_{\text{Hybrid}} \bowtie \text{EncIdeal}$$

```
log := [];
```

Alice =

```
let c_out = enc(K, 000000);
log[c_out] := N;
output c_out;
```

Bob =

```
recv c;
if let x = log[c] then
  let c_out = enc(K, 000000);
  log[c_out] := x + 1;
  output c_out;
else skip
```

Idealized Protocol

no dataflow from N to output!

$$P'_{\text{Ideal}}$$

```
log := [];
```

Alice =

```
let c_out = enc(K, 000000);
log[c_out] := 0000000;
output c_out;
```

Bob =

```
recv c;
if let x = log[c] then
  let c_out = enc(K, 000000);
  log[c_out] := x + 1;
  output c_out;
else skip
```

Idealized Protocol

no dataflow from N to output!

$$P_{\mathtt{Ideal}} \approx P'_{\mathtt{Ideal}}$$

Computational Noninterference:

Changing value of secret nonce N does not change external behavior

Proving Secrecy

$$\text{Goal}: \quad \forall A, \exists S, A \bowtie P_{\text{Real}} = S$$

$$\begin{aligned} A \bowtie P_{\text{Real}} &\approx A \bowtie (P_{\text{Hybrid}} \bowtie \text{EncReal}) \\ &\approx A \bowtie (P_{\text{Hybrid}} \bowtie \text{EncIdeal}) \\ &\approx A \bowtie P_{\text{Ideal}} \\ &\approx A \bowtie P'_{\text{Ideal}} \quad =: \text{S} \end{aligned}$$

Computational Provers

CryptoVerif:

automates security game rewrites

used for TLS, Kerberos, SSH, Signal, WireGuard, …

Squirrel:

hides complex rewrites behind symbolic-looking logic

Owl:

hides complex rewrites behind information flow type system

Computational Provers

EasyCrypt:

more expressive, based on Probabilistic Relational Hoare Logic

Today

- Symbolic Crypto: modelling crypto with abstract terms
- Computational Crypto: modelling crypto using indistinguishabilities

Next Time

A Comprehensive Symbolic Analysis of TLS 1.3

Cas Cremers
University of Oxford, UK

Marko Horvat
MPI-SWS, Germany

Jonathan Hoyland
Royal Holloway, University of London, UK

Sam Scott
Royal Holloway, University of London, UK

Thyla van der Merwe
Royal Holloway, University of London, UK

After that..

Computationally Sound Mechanized Proofs for Basic and Public-key Kerberos

B. Blanchet [*]
CNRS & École Normale Supérieure & INRIA
blanchet@di.ens.fr

A.D. Jaggard [†]
DIMACS
Rutgers University
adj@dimacs.rutgers.edu

A. Scedrov [‡]
Department of Mathematics
University of Pennsylvania
scedrov@math.upenn.edu

J.-K. Tsay [Σ]
Department of Mathematics
University of Pennsylvania
jetsay@math.upenn.edu