Computer and Network Security

Lecture 04: Confidentiality

COMP-5370/6370
Fall 2024

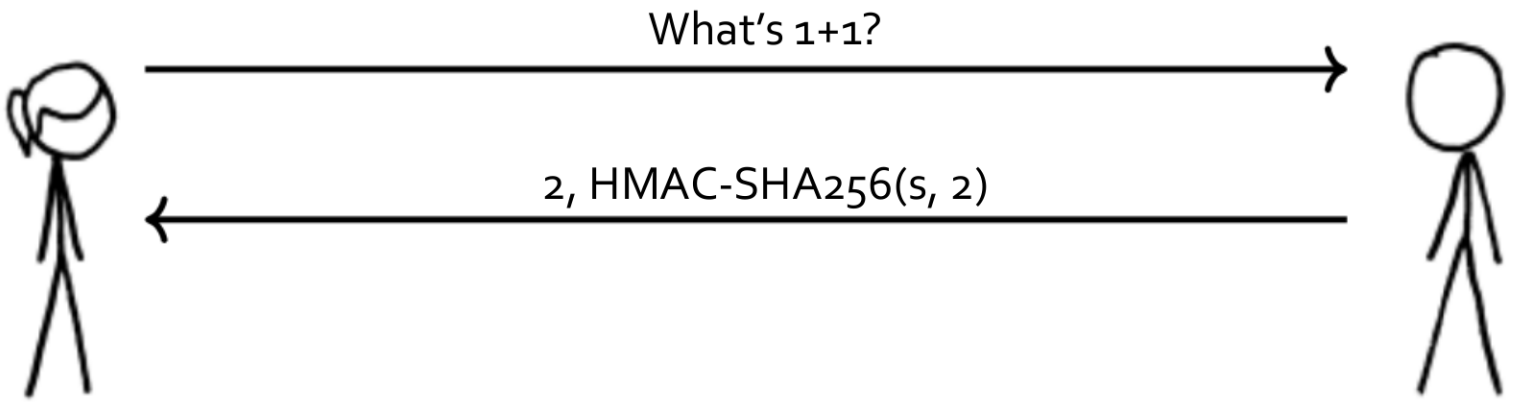
Properties of Secure Channel

A **secure channel** is a mechanism that allows Alice and Bob to communicate with the properties of:

- **Confidentiality**
  - Messages can't be read by a 3rd party (3P)
- **Message Integrity**
  - Messages can't be unknowingly modified by 3P
- **Sender Authenticity**
  - Valid messages creatable **only** by a 1P actor

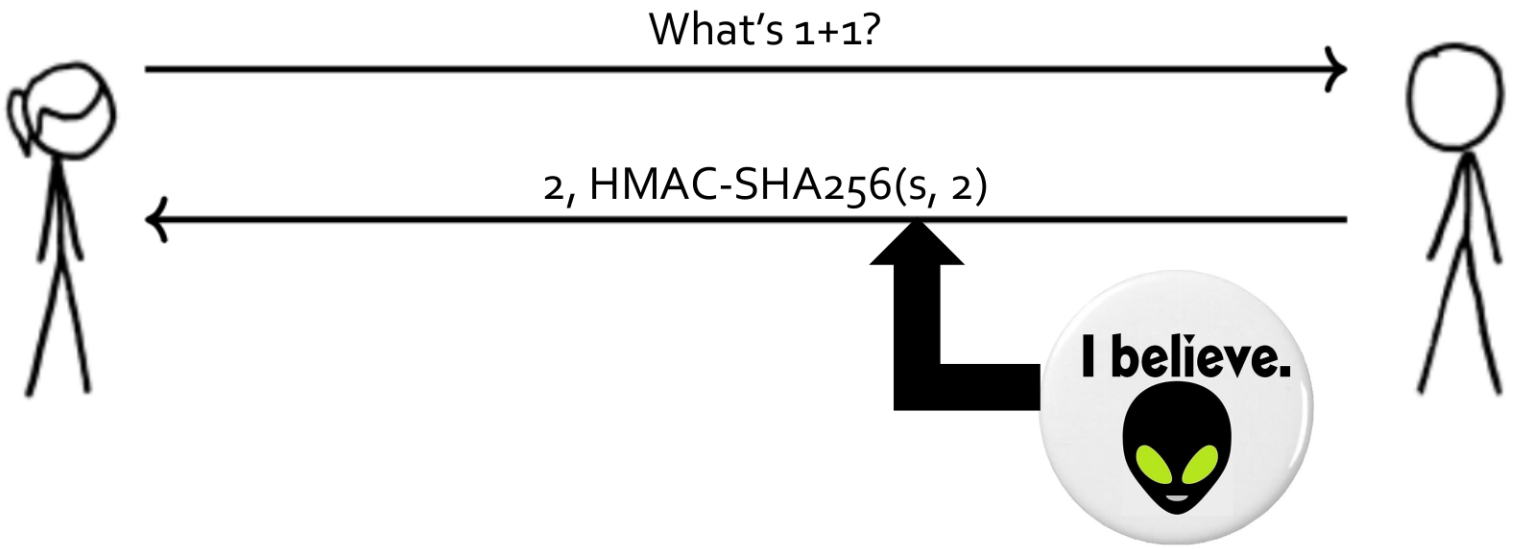
Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



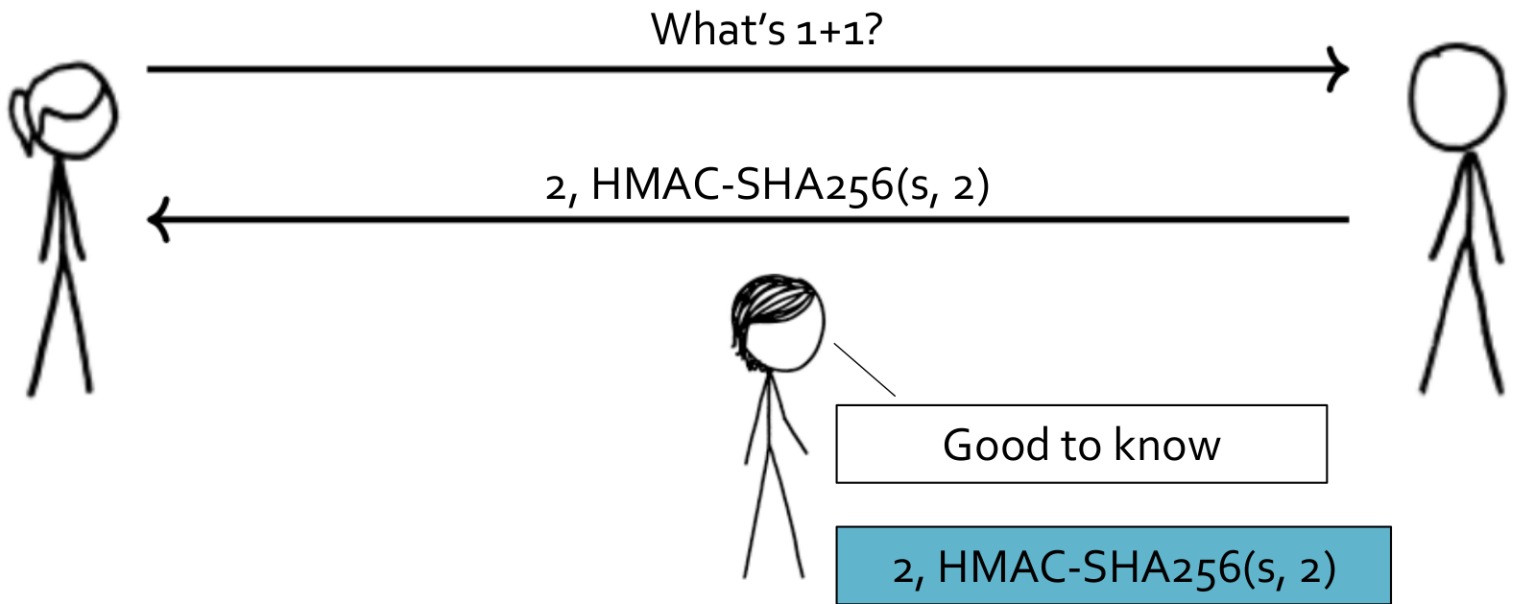
Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



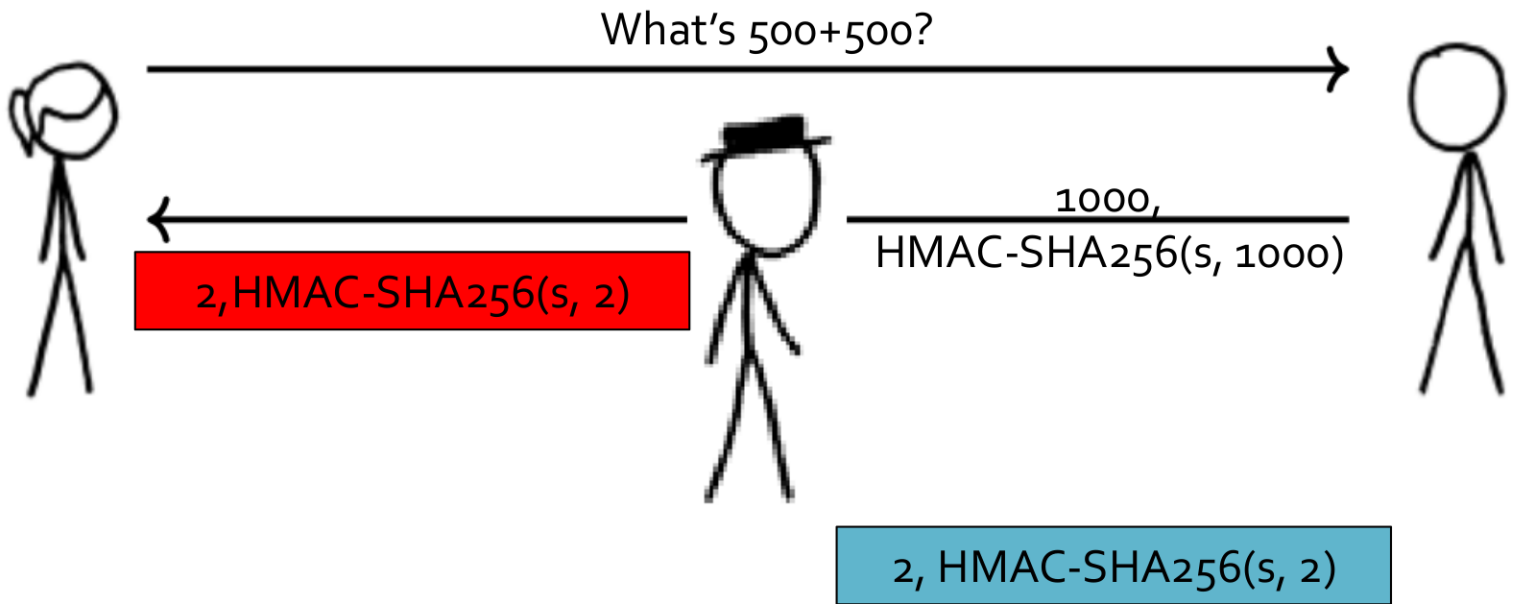
Replay Attacks

In our simple construction, using a MAC does **not** provide sender authenticity in the general case.



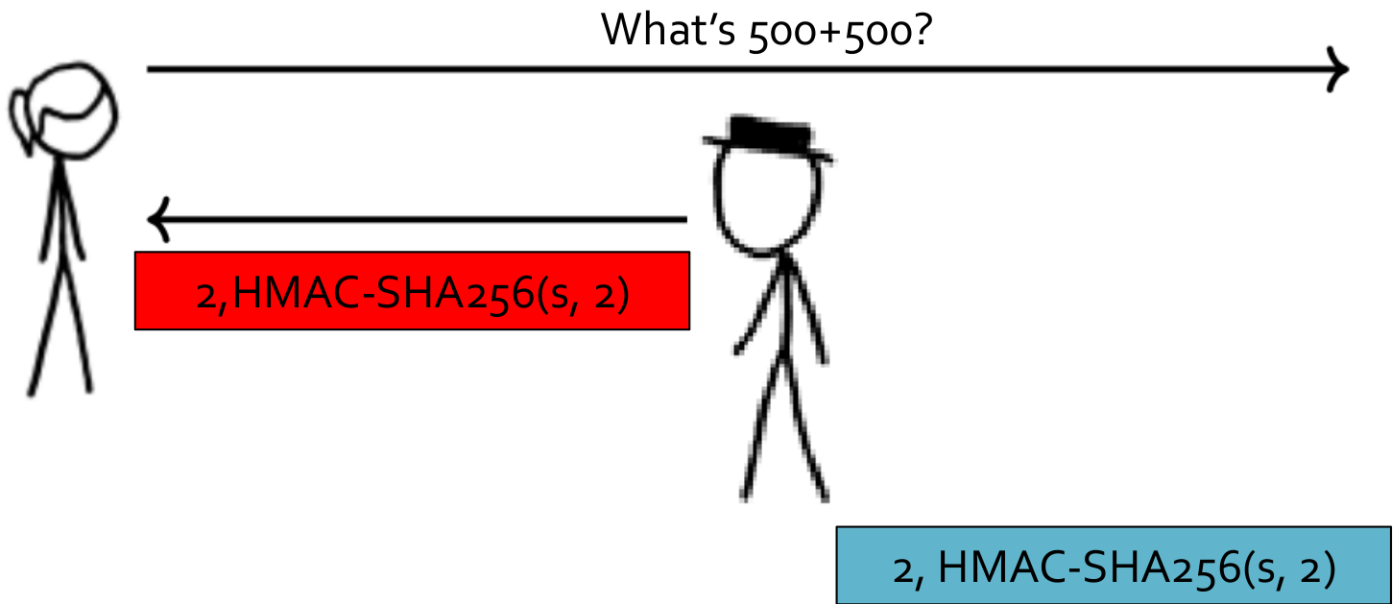
Replay Attacks

In our simple construction, using a MAC does **not** provide sender authenticity in the general case.



Replay Attacks

In our simple construction, using a MAC does **not** provide sender authenticity in the general case.



Freshness in Communications

An important aspect of S&P is ensuring not only sender authenticity, but also **freshness** of the messages being exchanged.

- Knowing **when** you're talking with them

WARNING

I AM NOT A CRYPTOGRAPHER

WARNING

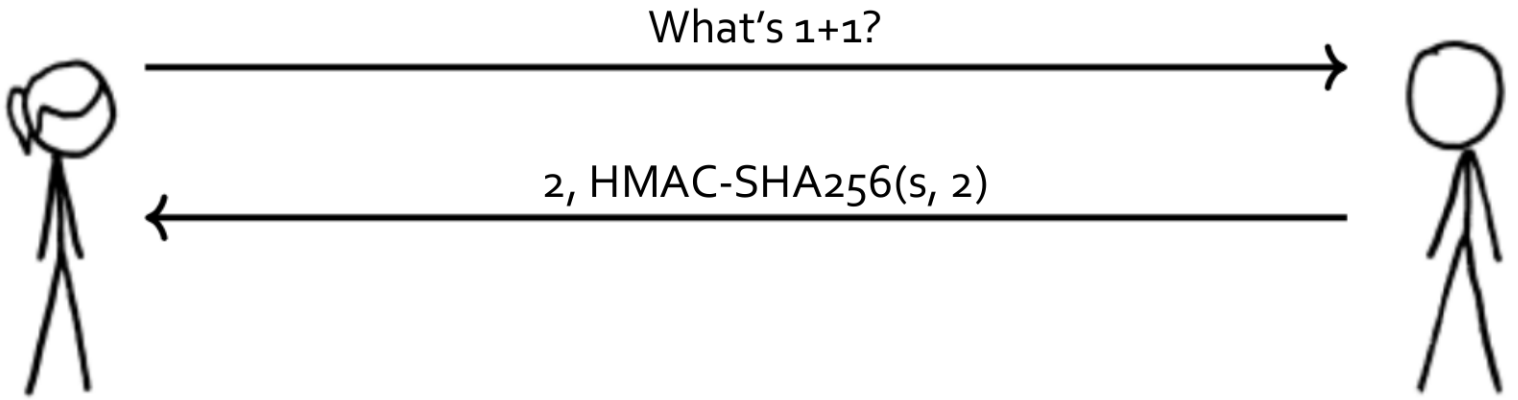
YOU ARE NOT A CRYPTOGRAPHER


THE FIRST RU
IS YOU DON'T ROLL
imgflip.com
THE SECOND RULE OF CRYPTO
IS DON'T ROLL YOUR OWN CRYPTO
imgflip.com


THE THIRD RULE OF CRYPTO
IS YOU DON'T ROLL YOUR OWN CRYPTO
memegenerator.net

Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



Properties of Secure Channel

A **secure channel** is a mechanism that allows Alice and Bob to communicate with the properties of:

- **Confidentiality**
  - Messages can't be read by a 3rd party (3P)
- **Message Integrity**
  - Messages can't be unknowingly modified by 3P
- **Sender Authenticity**
  - Valid messages creatable **only** by a 1P actor

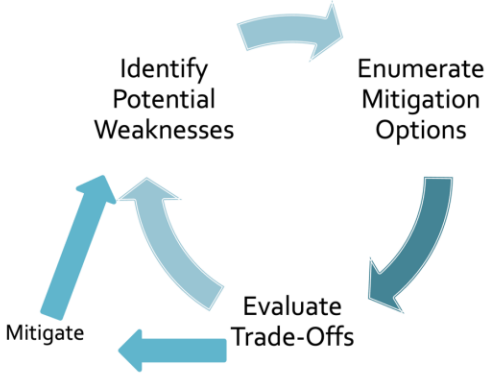
Thinking about Properties

Adversary

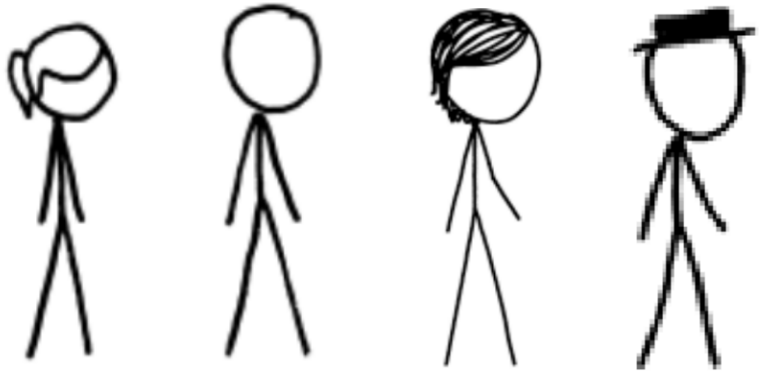
- Intelligent Actor
  - Person, Group, or Organization
- Have own:
  - Capabilities
  - Motivations
  - Intentions
- Are **NOT** restricted by expectations

Threat Modeling

A systematic approach to analyzing and understanding potential weaknesses.



For **confidentiality**, who should we be worried about?



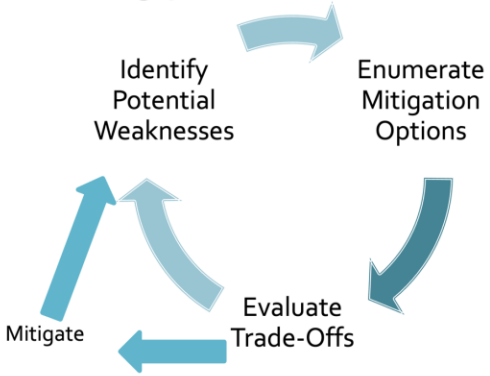
Thinking about Properties

Adversary

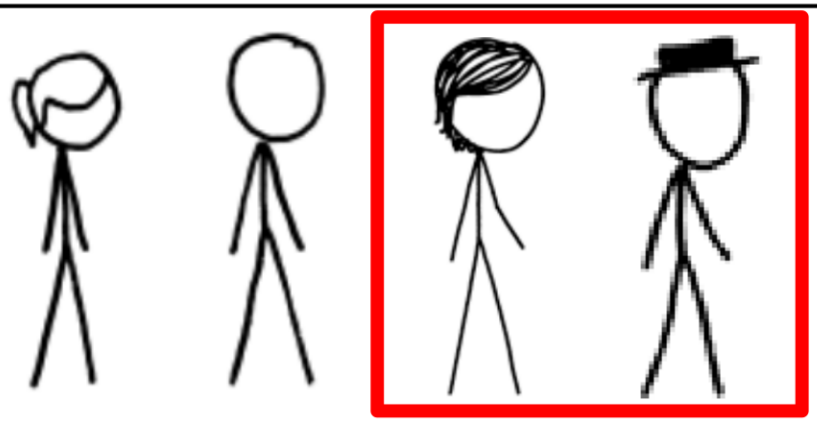
- Intelligent Actor
  - Person, Group, or Organization
- Have own:
  - Capabilities
  - Motivations
  - Intentions
- Are **NOT** restricted by expectations

Threat Modeling

A systematic approach to analyzing and understanding potential weaknesses.



For **confidentiality**, who should we be worried about?



Malicious Mallory

- An **active** and malicious actor
  - Has all passive capabilities (read messages)
  - Can modify/delay/discard messages
  - Can be an unintended end-point (MitM attack)

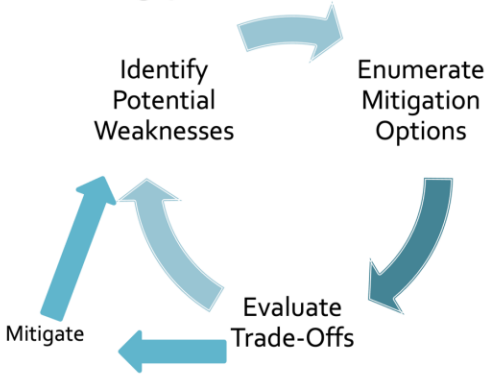
Thinking about Properties

Adversary

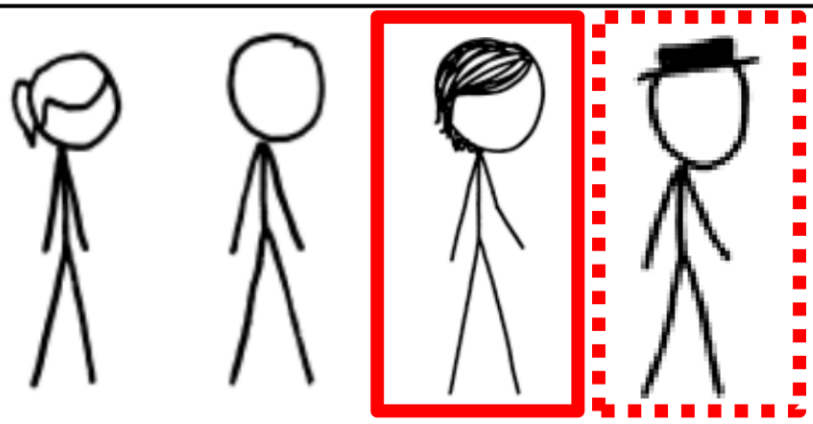
- Intelligent Actor
  - Person, Group, or Organization
- Have own:
  - Capabilities
  - Motivations
  - Intentions
- Are **NOT** restricted by expectations

Threat Modeling

A systematic approach to analyzing and understanding potential weaknesses.

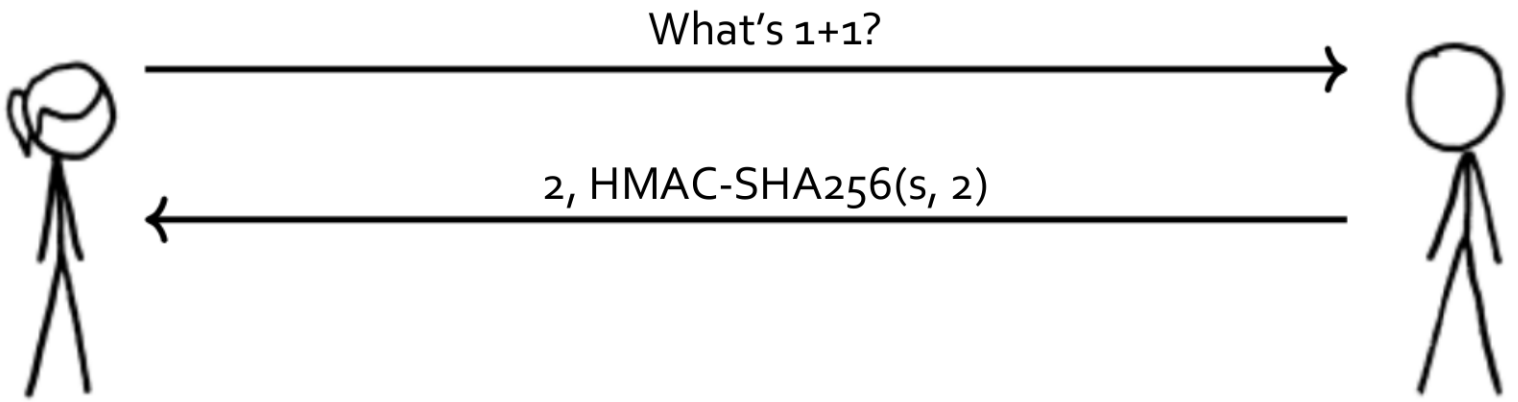


For **confidentiality**, who should we be worried about?



Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



Terminology

Plaintext (PT) – Unencrypted message that is "readable" message to everyone.

Ciphertext (CT) – Encrypted message that is "opaque" to everyone.

Cipher – Algorithm used for encrypting (PT→CT) and decrypting (CT→PT).

Key Material (key) – Instance-specific secret required to operate the cipher in a useful way.

Anyone with key can encrypt and decrypt.

Kerckhoffs's Principle

A cryptosystem should remain secure even if 100% of the system is publicly-known except the key material.

Kerckhoffs's Principle (Break-Out #1)

Implementation shouldn't rely on "security through obscurity".

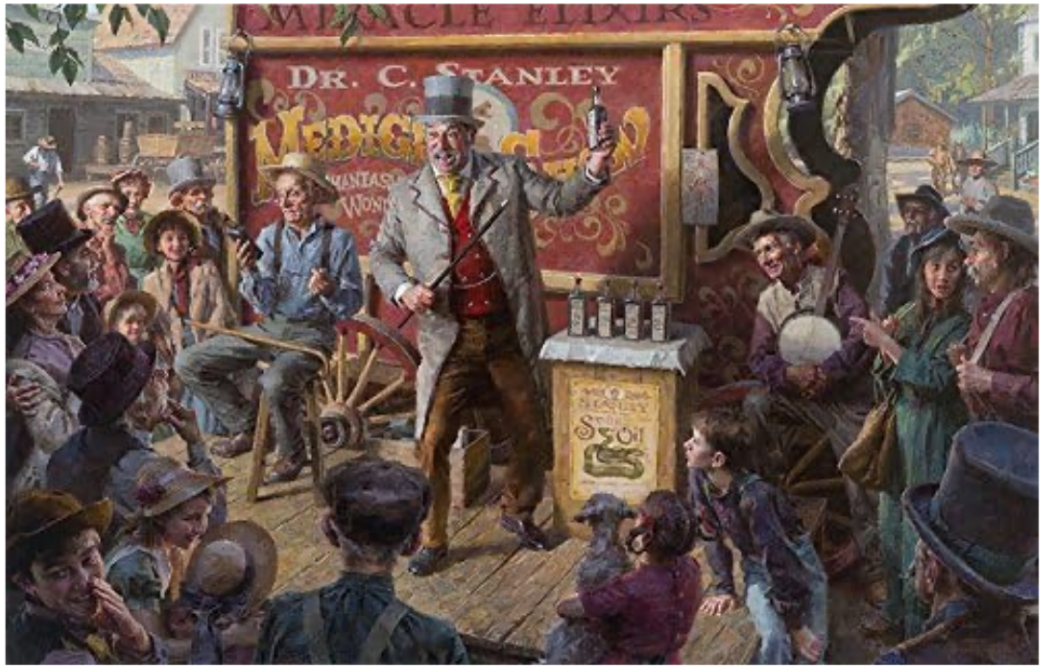
Kerckhoffs's Principle (Break-Out #1)

Implementation shouldn't rely on "security through obscurity".

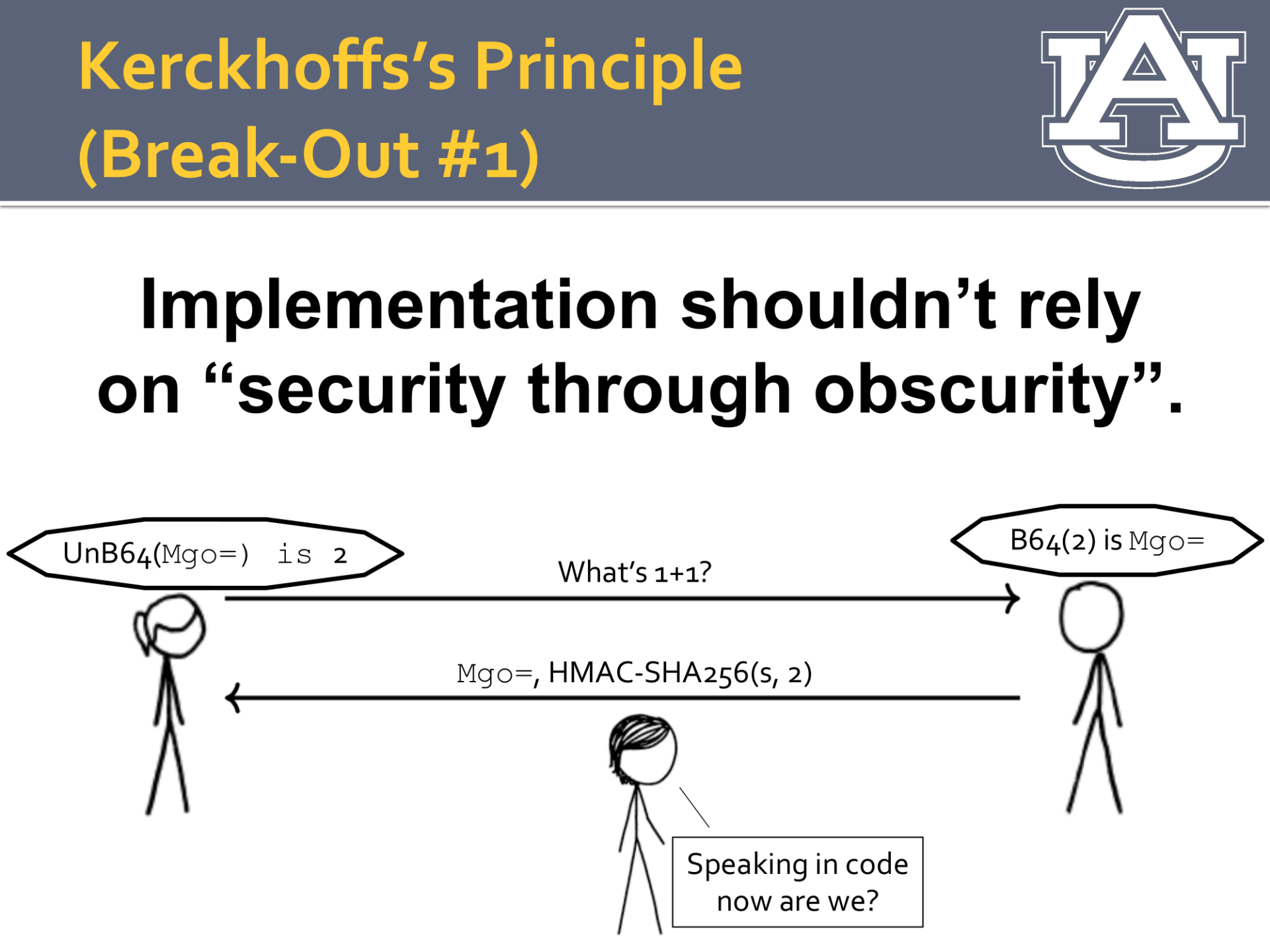
A system/component is explicitly and knowingly reliant on the non-availability of info about the design/architecture.

Obfuscation

Obfuscation when messages are “munged” in a way as to appear to be safe but without adding any actual security.



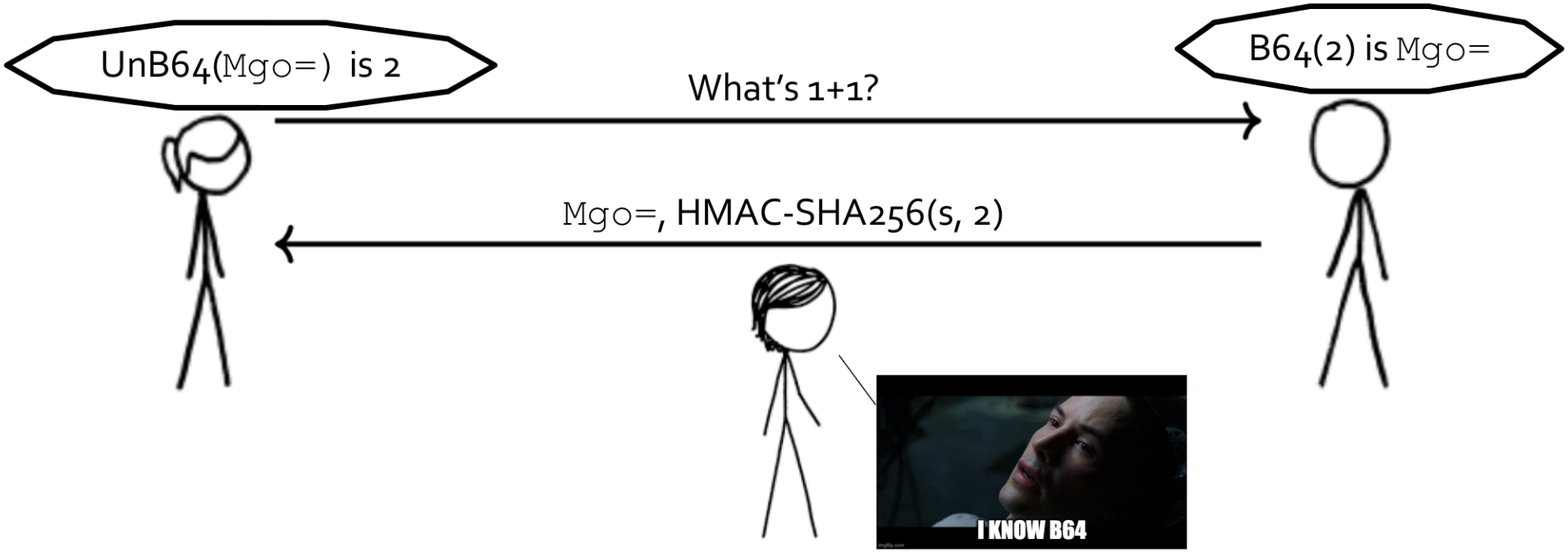
Kerckhoffs's Principle (Break-Out #1)



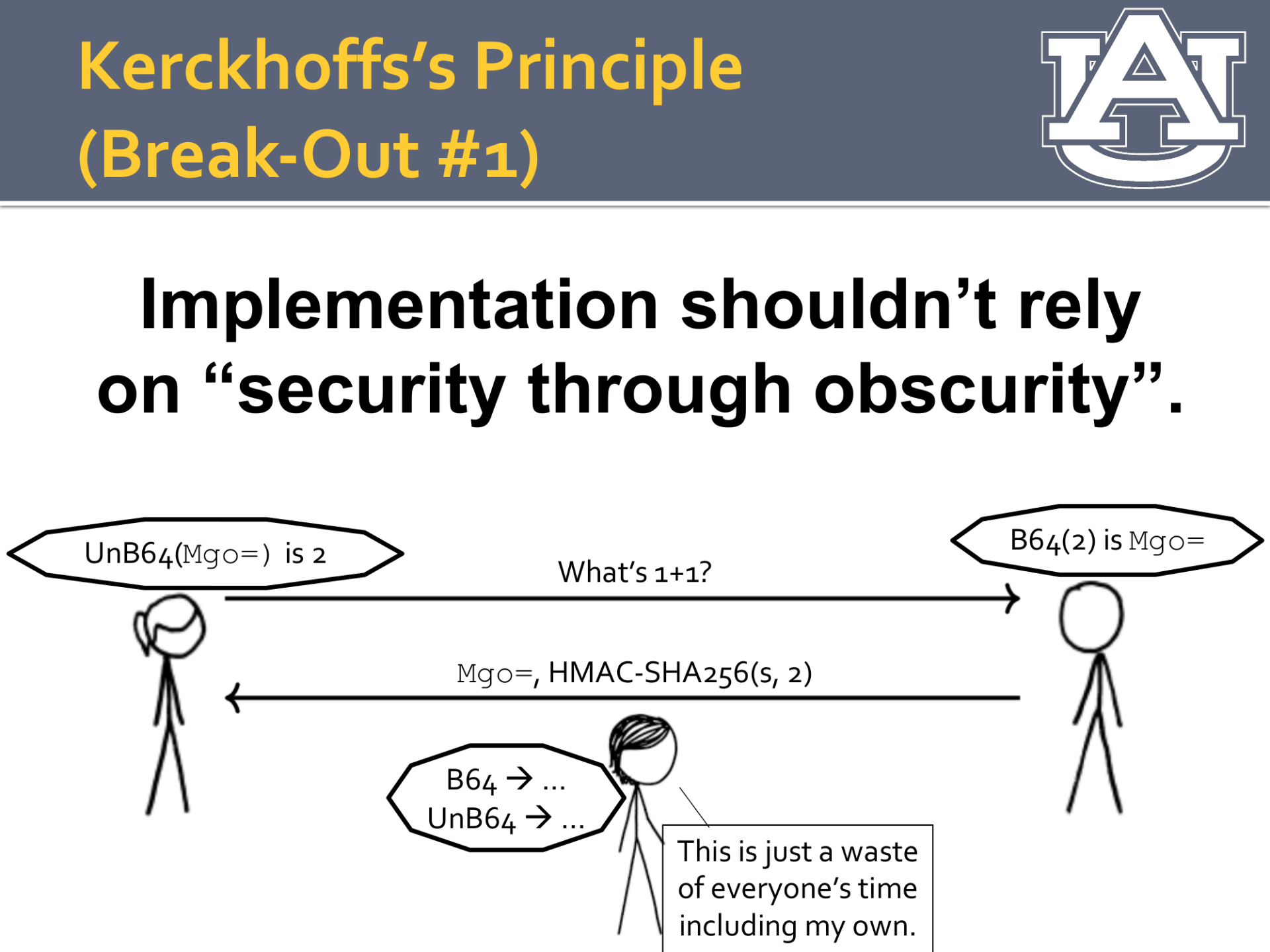
Implementation shouldn't rely on "security through obscurity".

Kerckhoffs's Principle (Break-Out #1)

Implementation shouldn't rely on "security through obscurity".



Kerckhoffs's Principle (Break-Out #1)



Implementation shouldn't rely on "security through obscurity".

Kerckhoffs's Principle (Break-Out #2)

Ciphertext must be realistically secure at a minimum but provably secure if possible.

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

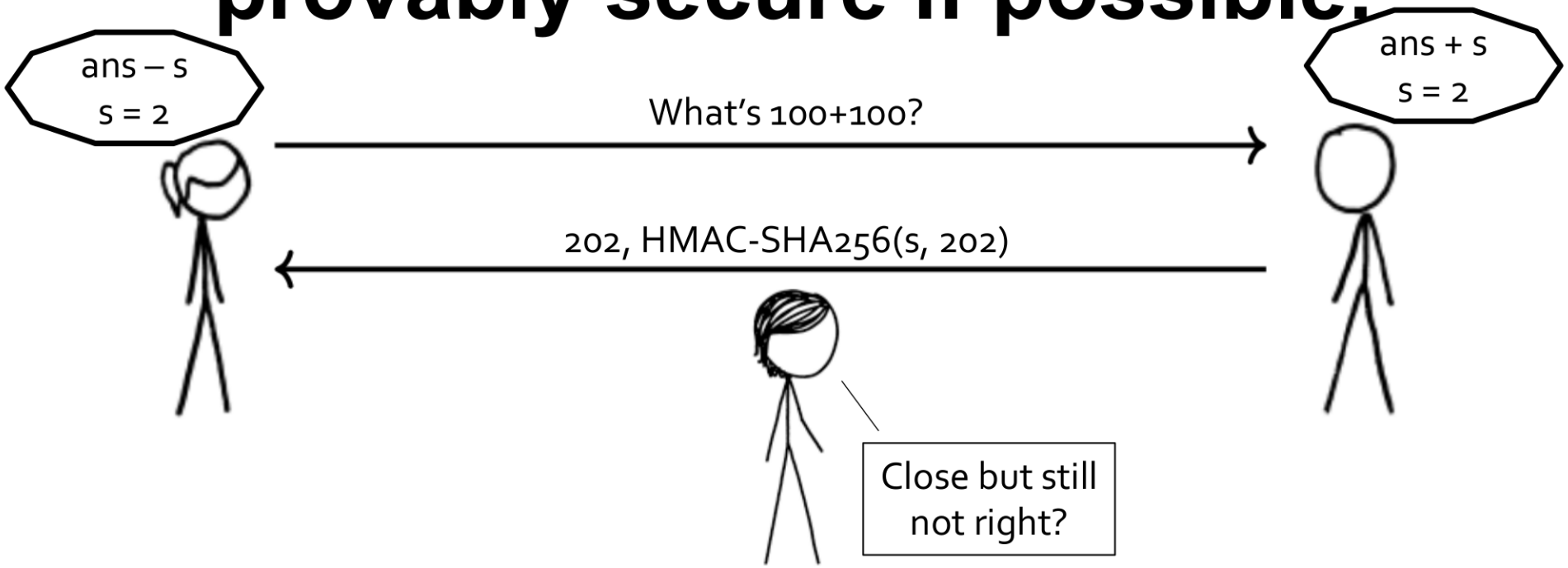
$$Sim((y_0, y_1), c) := ((t_0, t_1), (z_0, z_1)),$$

where

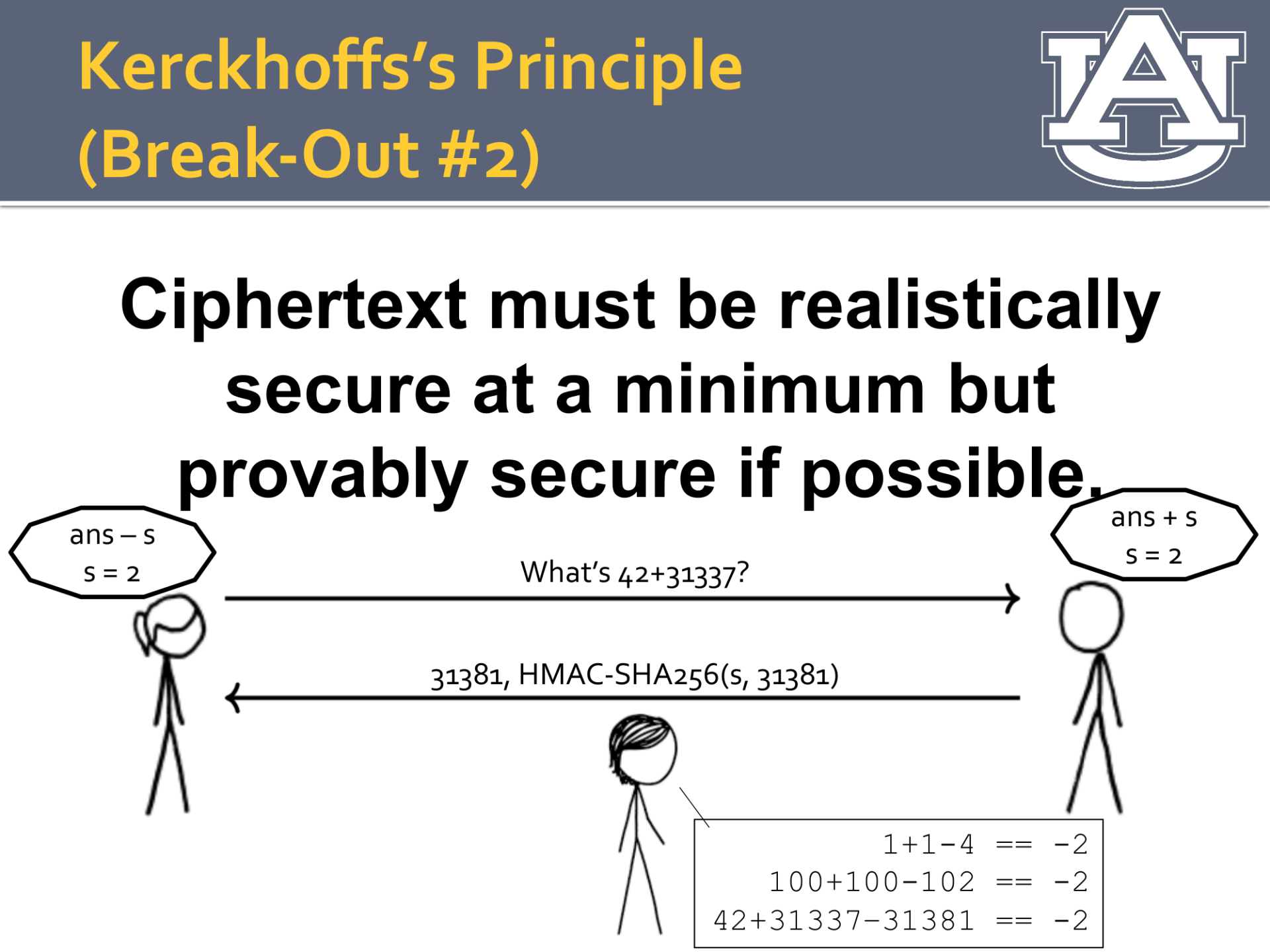
$$(t_0, z_0) \xleftarrow{R} Sim_0(y_0, c) \quad \text{and} \quad (t_1, z_1) \xleftarrow{R} Sim_1(y_1, c).$$

Kerckhoffs's Principle (Break-Out #2)

Ciphertext must be realistically secure at a minimum but provably secure if possible.



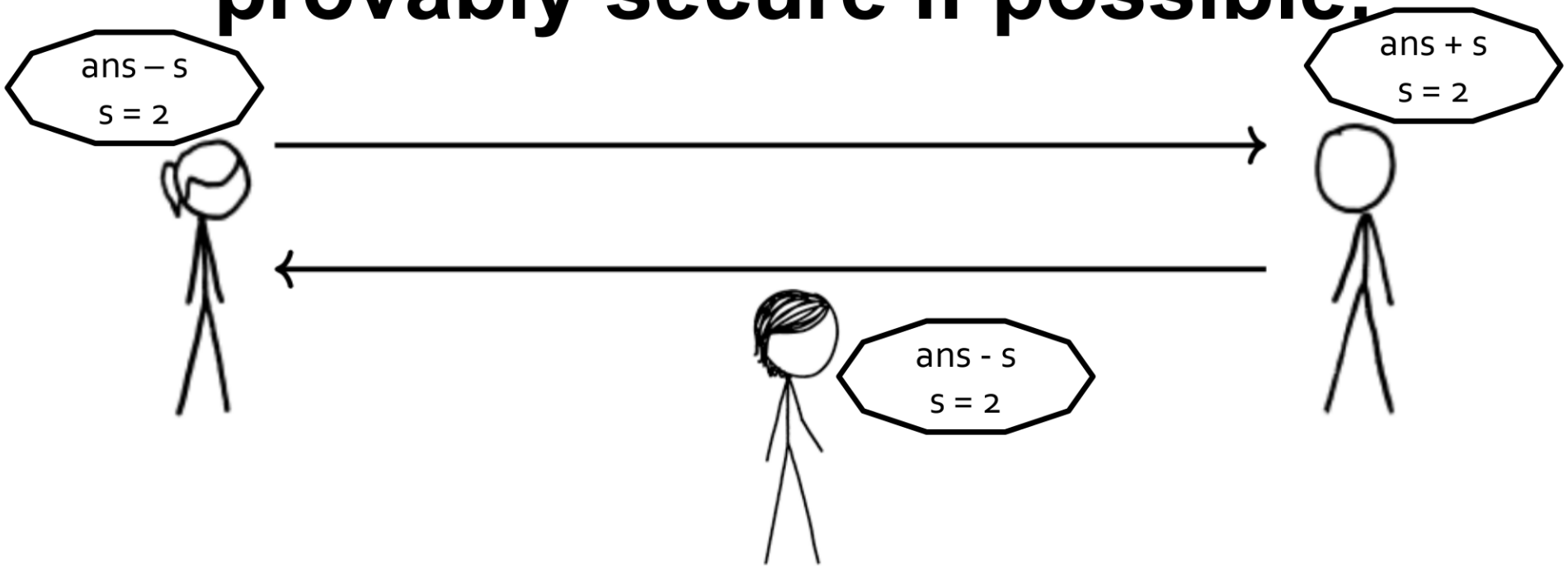
Kerckhoffs's Principle (Break-Out #2)



Ciphertext must be realistically secure at a minimum but provably secure if possible.

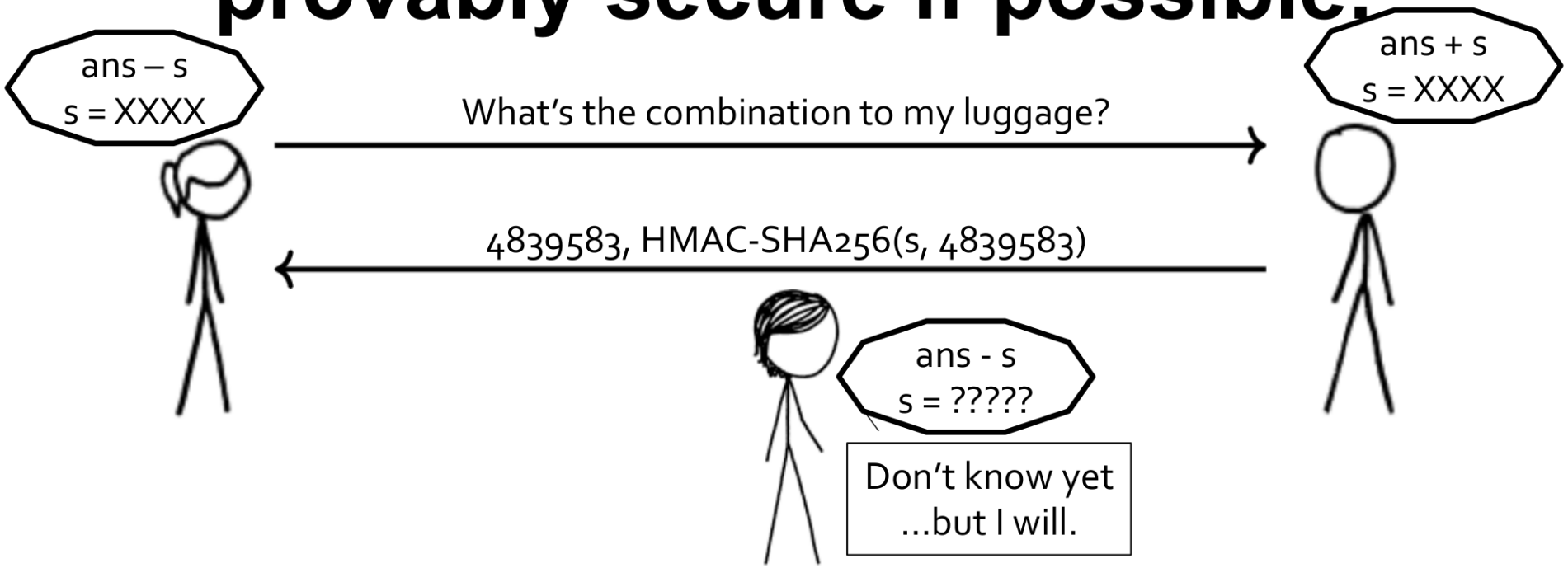
Kerckhoffs's Principle (Break-Out #2)

Ciphertext must be realistically secure at a minimum but provably secure if possible.



Kerckhoffs's Principle (Break-Out #2)

Ciphertext must be realistically secure at a minimum but provably secure if possible.



Kerckhoffs's Principle (Break-Out #3)

Key material must be easy to change, verify, store, and transfer.

Key Rotation

Key Rotation is the process of replacing in-use key material for all 1P actors without greatly interfering with operation.

- Static keys are bad (non-rotatable)
- Rotatable keys are good (can be rotated)
- Rotating keys are best (are being rotated)

- Using the **right** key at the **right** time is just as important as **having** the key.

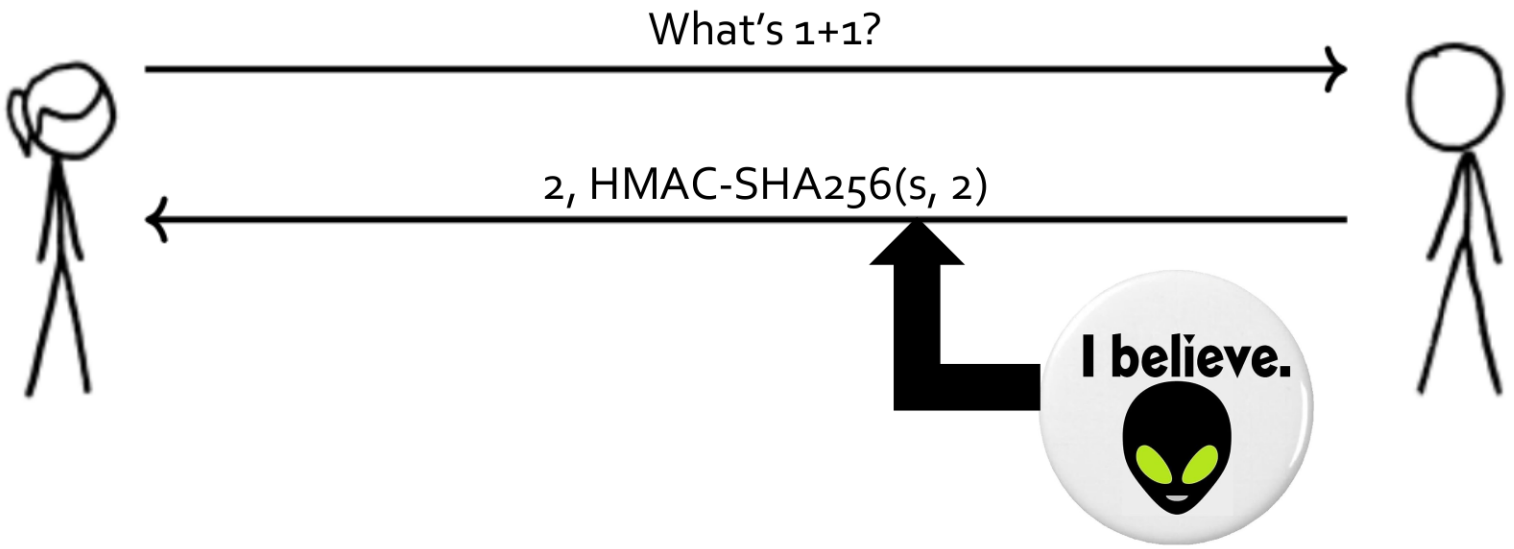
Kerckhoffs's Principle (Break-Out #3)

Key material must be easy to change, verify, store, and transfer.

- Change: Rotate
- Verify: Check correctness
- Store: Hold until later
- Transfer: Deliver new versions

Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity

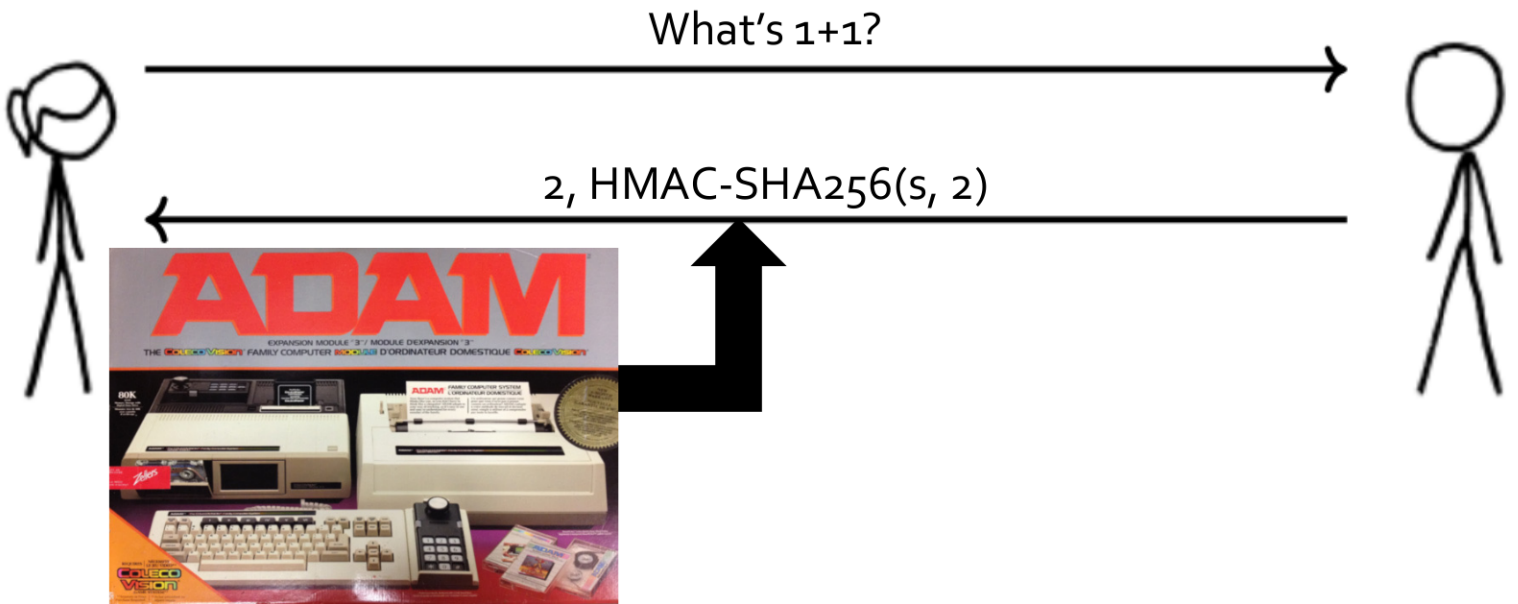


Kerckhoffs's Principle (Break-Out #4)

Don't make unrealistic assumptions about the abilities or competence of the humans who use/operate the system.

Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



HMAC-SHA256 by Hand

Kerckhoffs's Principle (Break-Out # 4)

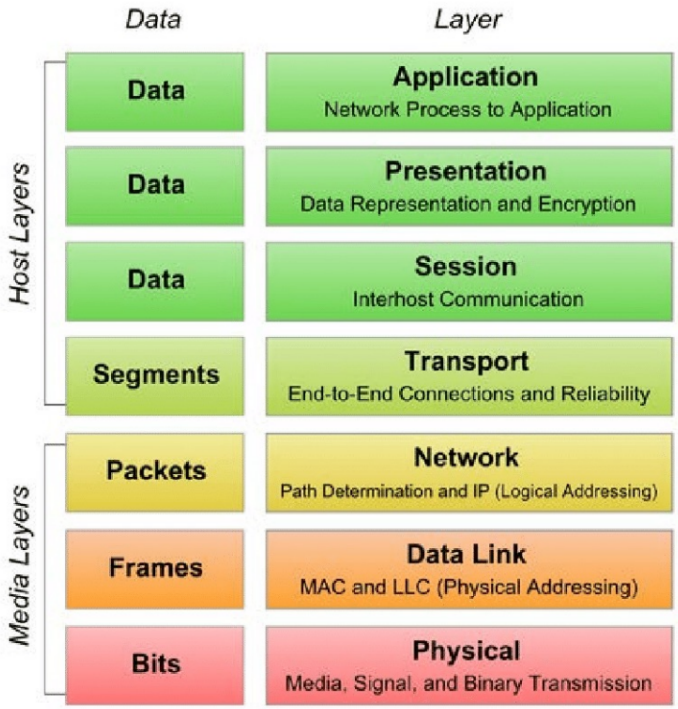
Don't make unrealistic assumptions about the abilities or competence of the humans who use/operate the system.

- Humans are an unsolved security problem
  - Will fail in every known, imaginable, and unimaginable way possible.
  - Sometimes will fail intentionally but w/o malice
- Humans may be an unsolvable problem

Kerckhoffs's Principle (Break-Out #5)

Interoperate with existing infrastructures, topologies, and protocols at higher and lower levels

- Ideally, system should be 100% transparent to existing infrastructure
- **Systems that are hard to deploy usually don't get deployed**



Kerckhoffs's Principle (Break-Out #6)

Should be generic and reusable across many different hardware and software platforms.

- Reusable and re-implementable
- Over customization for a single use-case reduces ability to be leveraged for others

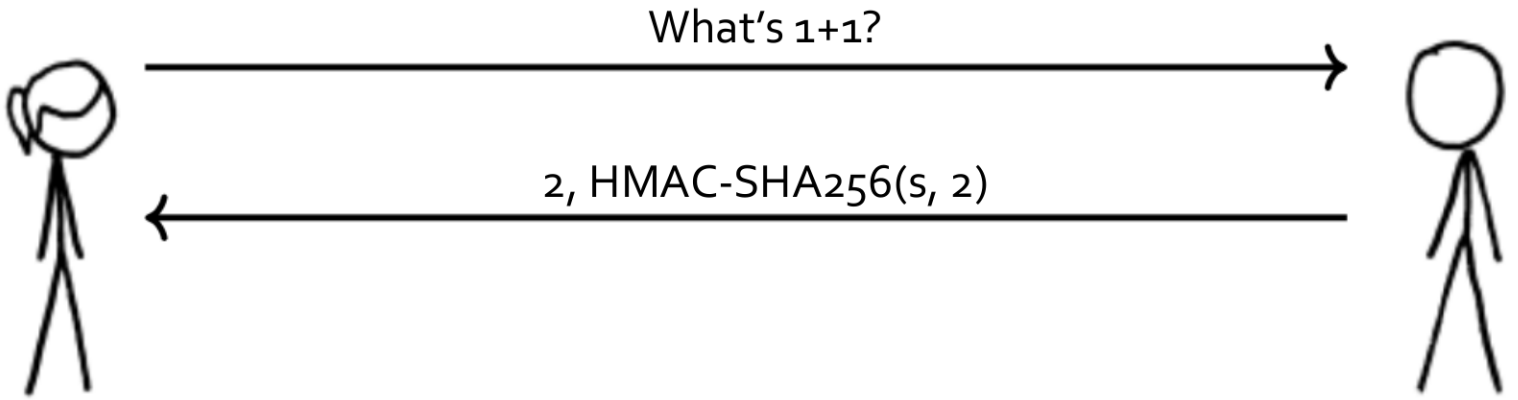
Kerckhoffs's Principle

A crypt should
remai even if
100% tem is
public except
the rial.

La Cryptographie Militaire (1883)

Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



One-Time Pad

One-Time Pad is the only cryptosystem known to be unbreakable even infinite computational resources.

- `ct[i] = pt[i] XOR key[i]`
- Extremely fast to encrypt and decrypt
- Extremely easy to implement safely

One-Time Pad Example

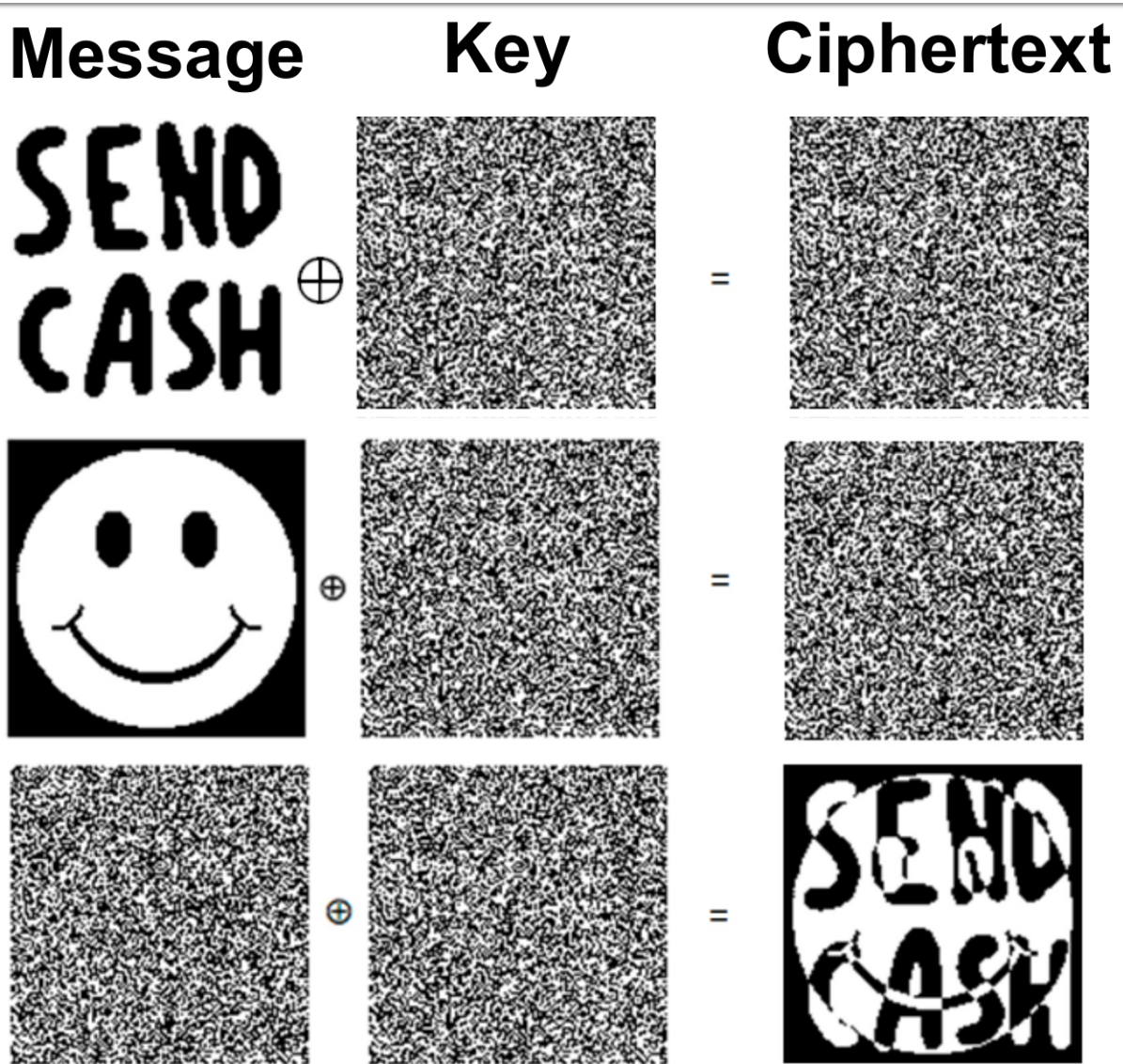
Encryption

plaintext		01101010110100 1100
key	XOR	11010101010110 1011
ciphertext		10111111100010 0111

Decryption

ciphertext		10111111100010 0111
key	XOR	11010101010110 1011
plaintext		01101010110100 1100

N-Time Pad Leaks Information



One-Time Pad Keying

```
ct[i] = pt[i] XOR key[i]
```

Due to the requirement that the key never be reused, naïve one-time pad ciphers require:

- Strong randomness for all key material
- Message-length key material
- Per-message key material

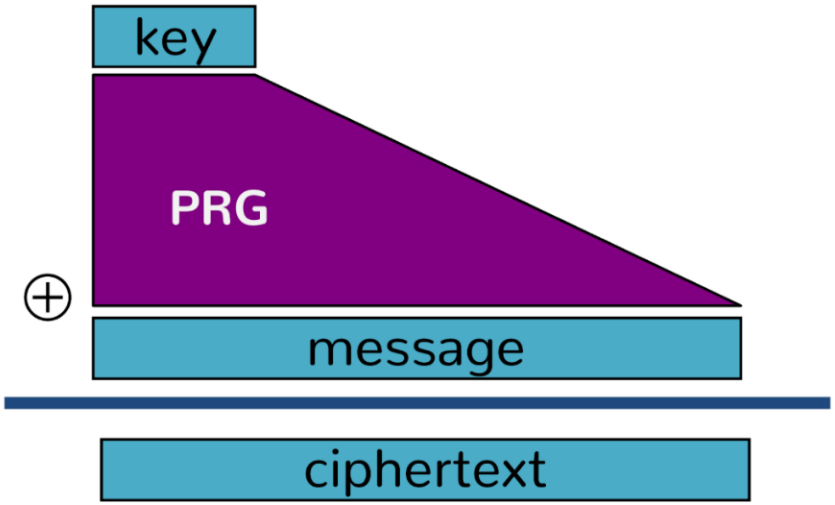
Lots and Lots of KEYMAT

Pseudorandom Number Generator (PRNG)

A **Pseudorandom Number Generator (PRNG)** maps a k-bit random input to an n-bit pseudorandom output (n > k).

- Used to “expand” randomness into more random-like data
- Use a secret “seed” (s) for unpredictability

Stream Cipher



- Shared seed known by all participants
- Seed is “expanded” to the length of the message
  - PRNG

Infinite-Length One-Time Pad

Pseudorandom Number Generator (PRNG)

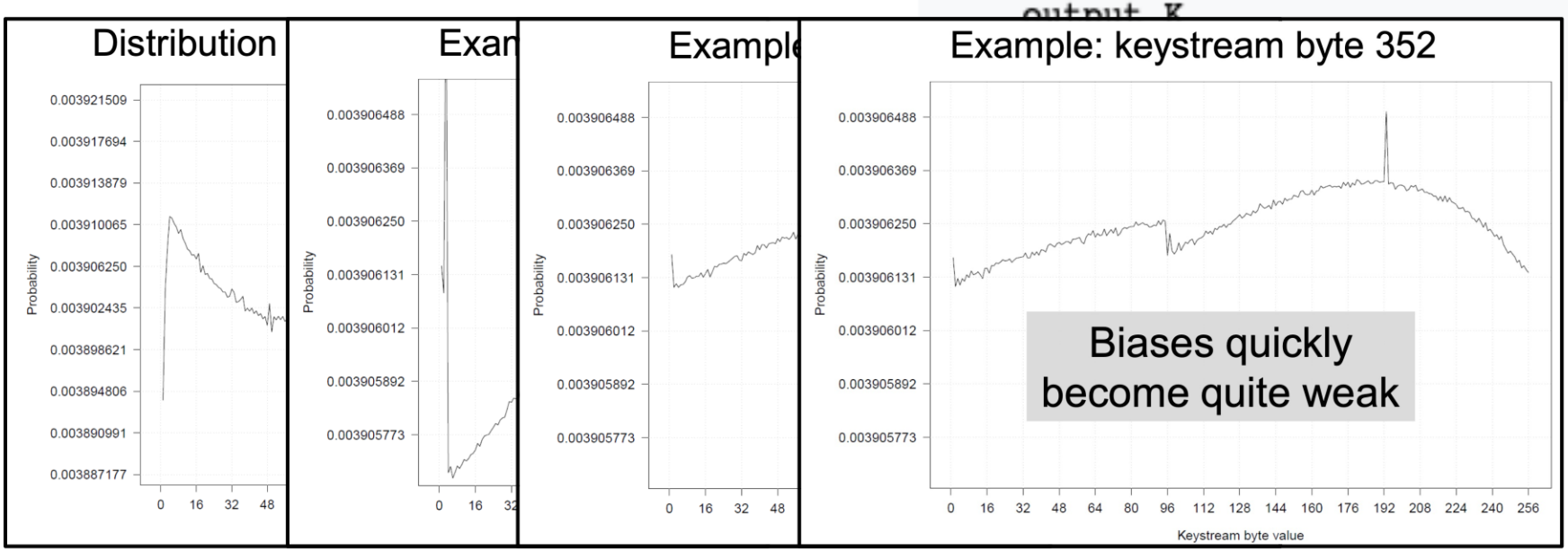
A **Pseudorandom Number Generator (PRNG)** maps a k-bit random input to an n-bit pseudorandom output (n > k).

- Used to "expand" randomness into more random-like data
- Use a secret "seed" (s) for unpredictability
- **Not safe for generating keys**
- **Safe for some uses crypto usage but only *SOME* uses**

RC4 Stream Cipher

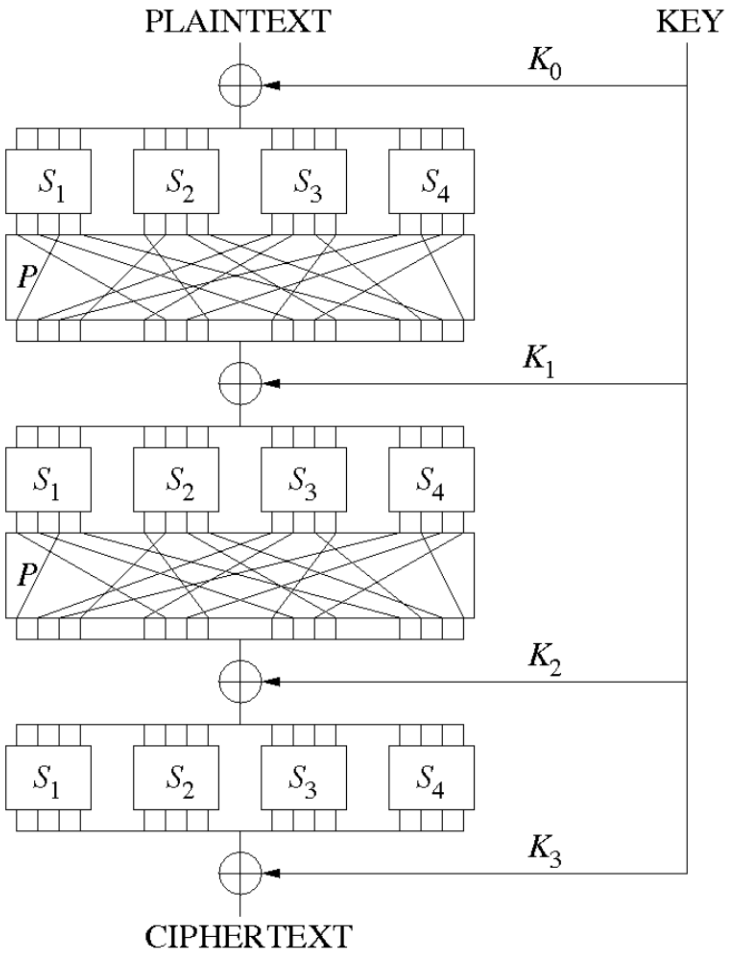
- Was widely used for speed and simplicity
- Should **not** be used

```
i := 0
j := 0
while GeneratingOutput:
    i := (i + 1) mod 256
    j := (j + S[i]) mod 256
    swap values of S[i] and S[j]
    K := S[(S[i] + S[j]) mod 256]
    output K
```



https://www.rc4nomore.com/

Block Cipher



- Fixed-size input
- Fixed-size output
- Substitutions from secret internal state
  - "S-Boxes"
- Multiple "rounds" to increase substitutions

DES – Data Encryption Standard

- 1977 – Standardized by NIST
  - NSA heavily involved in design
- 64-bit block cipher using 56-bit key
- Often implemented in hardware due to computation needs and complexity

- 1990 – Differential cryptanalysis discovered
  - General technique against block ciphers
- 1998 – EFF DES Cracker operational
  - Brute-force attack on key

DES – Data Encryption Standard

Never ever, ever, ever use single-DES

3DES – Triple DES

- 1995 – A “hot patch” for DES via RFC
- Exact same algorithm w/ multiple keys
  - Encrypt → decrypt → encrypt
- Best-case construction is 168-bit key
- Vulnerable to “meet-in-the-middle” attacks
  - Brute-force: 2^{56} space + 2^{112} operations
- 2016 – Practical collision attack (Sweet32)
  - DES is 64-bit block cipher ($2^{36.6}$ blocks needed)
  - “Got lucky” w/ 2^{20} block in 25 minutes vs. TLS

3DES – Triple DES

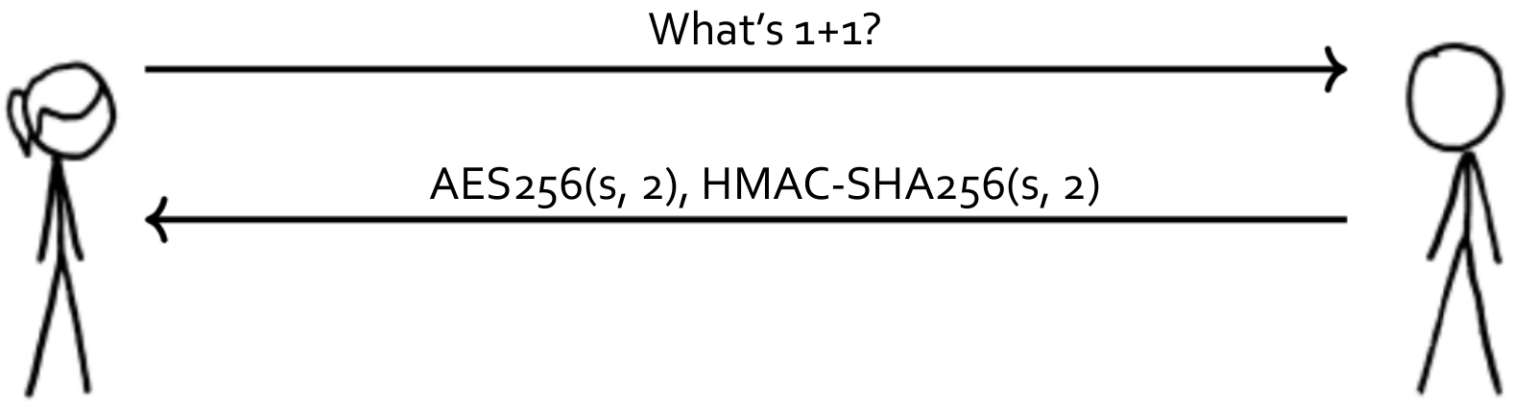
3DES is a weak cipher and should be immediately deprecated.

AES – Advanced Encryption Std

- 2001 – Standardized by NIST
- 128-bit block size
- 128/192/256-bit keys
  - Bigger key → same algorithm + more rounds
- Invertible S-boxes
  - Same used for both Encrypt() and Decrypt()
- AES-256 approved for CNSA
  - "Commercial National Security Algorithm Suite"
  - Encrypt TOP SECRET information and broadcast

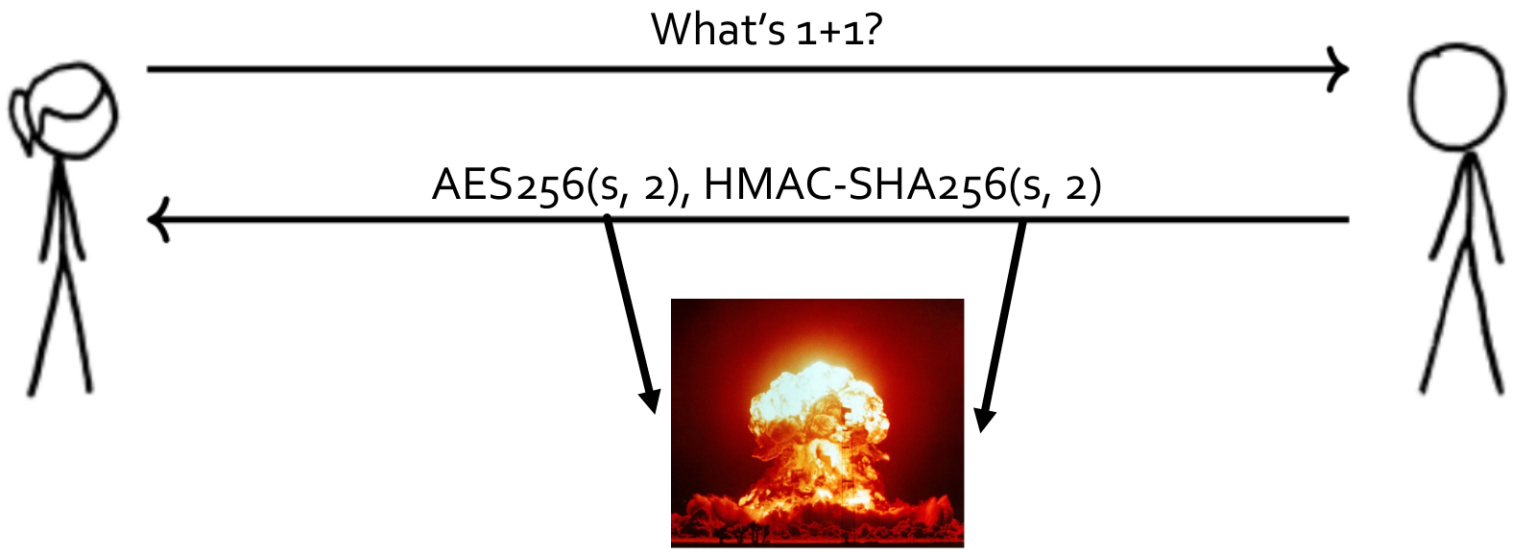
Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



Problem 1

Re-using key material for different algorithms can reveal information about the key material's value.



Key Derivation Function (KDF)

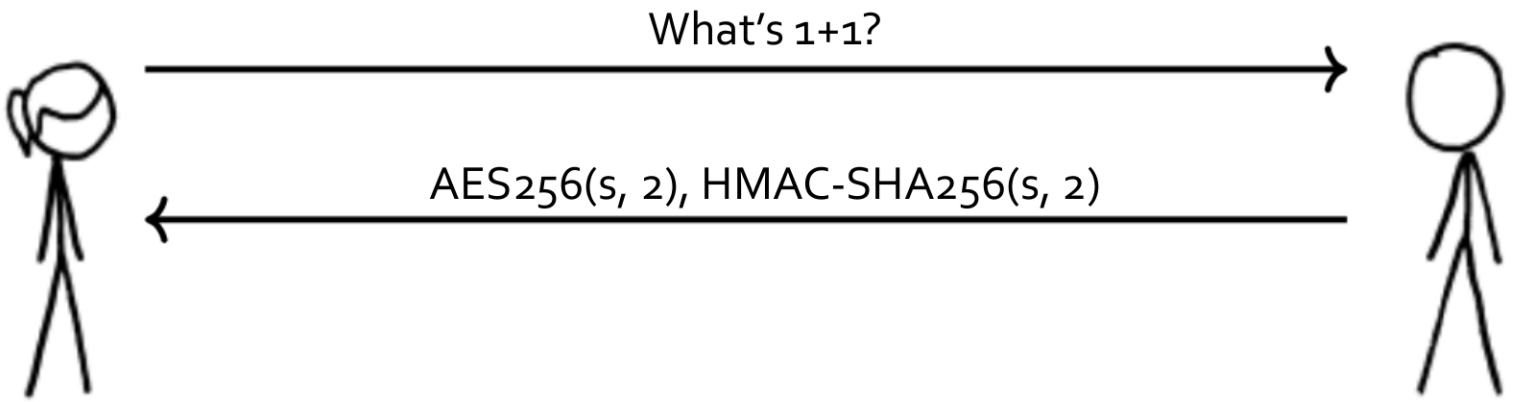
A **Key Derivation Function (KDF)** is one which can *safely* and deterministically turn one shared-secret into multiple.

- 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", "")
```

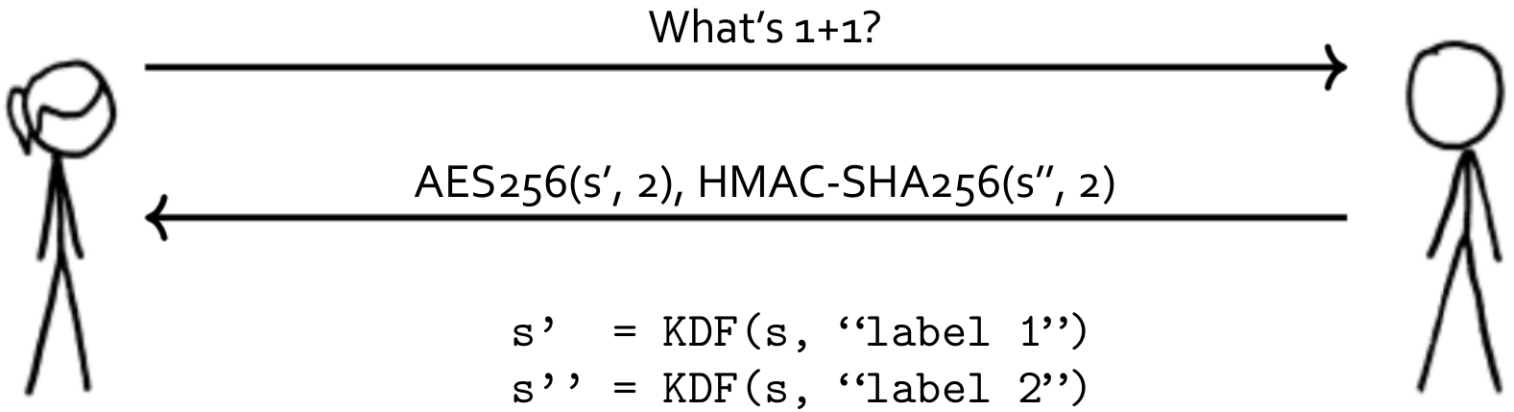
Building a Secure Channel

Confidentiality
Message Integrity
Sender Authenticity



Building a Secure Channel

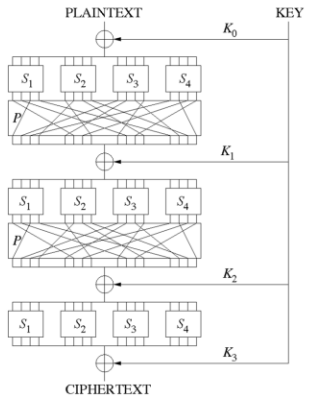
Confidentiality
Message Integrity
Sender Authenticity



Problem 2

Block ciphers are fixed-length inputs/outputs and messages are … not.

Block Cipher



- Fixed-size input
- Fixed-size output
- Substitutions from secret internal state
  - "S-Boxes"
- Multiple "rounds" to increase substitutions

Cipher Mode

A **cipher mode** is a way to use a fixed-size block cipher with arbitrary-sized data.

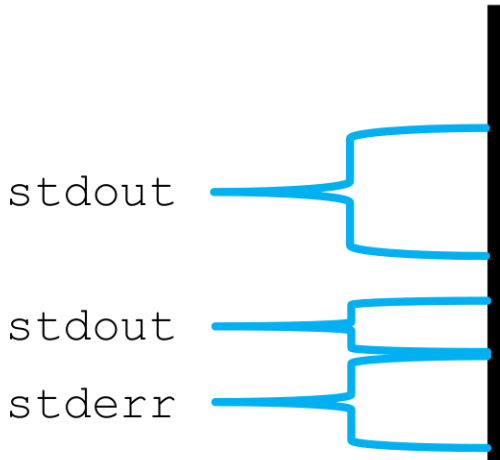
- Needed due to small/fixed cipher-width (AES256 == 256 bit blocks)
- **Choice can heavily impact the performance of the cryptosystem**

Computer and Network Security

Lecture 04: Confidentiality

COMP-5370/6370
Fall 2024

Project 1A Output




```
$>
$> make run FILE=./spec-testcases/valid/0001.input
begin-map
a -- num -- -6
end-map
$> make run FILE=./spec-testcases/invalid/0001.input
begin-map
ERROR -- invalid key-value split
make: *** [run] Error 66
$>
```

```
$>
$> make run FILE=./spec-testcases/invalid/0001.input 1> this-is-stdout 2> this-is-stderr
$>
$> cat this-is-stdout
begin-map
$>
$>
$> cat this-is-stderr
ERROR -- invalid key-value split
make: *** [run] Error 66
$>
```

You exit w/ code 66. Make converts to stderr message.

Project 1A Auto-Runner

- Very similar to the auto-grader but **is not** the auto-grader
- Runs only the spec's testcases
- If you haven't built your own testing harness, can be extended

```
$> python3 auto-runner.py
OK -- spec-testcases/valid/0001.input
OK -- spec-testcases/valid/0002.input
OK -- spec-testcases/valid/0003.input
OK -- spec-testcases/valid/0004.input
OK -- spec-testcases/valid/0005.input
OK -- spec-testcases/valid/0006.input
OK -- spec-testcases/valid/0007.input
OK -- spec-testcases/valid/0008.input
OK -- spec-testcases/valid/0009.input
OK -- spec-testcases/valid/0010.input
OK -- spec-testcases/valid/0011.input
OK -- spec-testcases/invalid/0001.input
OK -- spec-testcases/invalid/0002.input
OK -- spec-testcases/invalid/0003.input
$>
```

"But it looks the identical"

```
$>
$> cat ./spec-testcases/valid/0006.input
(<a:ef%00gh>)
$>
$> cat ./spec-testcases/valid/0006.output
begin-map
a -- string -- efgh
end-map
$>
$> cat ./spec-testcases/valid/0006.output | xxd
00000000: 6265 6769 6e2d 6d61 700a 6120 2d2d 2073  begin-map.a -- s
00000010: 7472 696e 6720 2d2d 2065 6600 6768 0a65  tring -- ef.gh.e
00000020: 6e64 2d6d 6170 0a                        nd-map.
$>
```

Computer and Network Security

Lecture 04: Confidentiality

COMP-5370/6370
Fall 2024