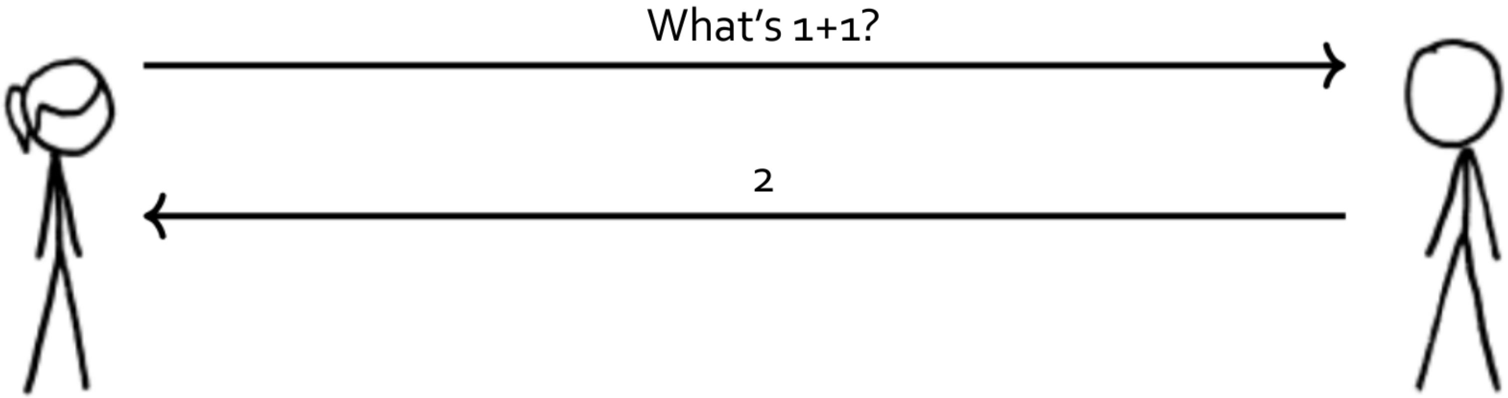
Computer and Network Security

Lecture 03:
Hashing and Integrity

COMP-5370/6370
Fall 2024

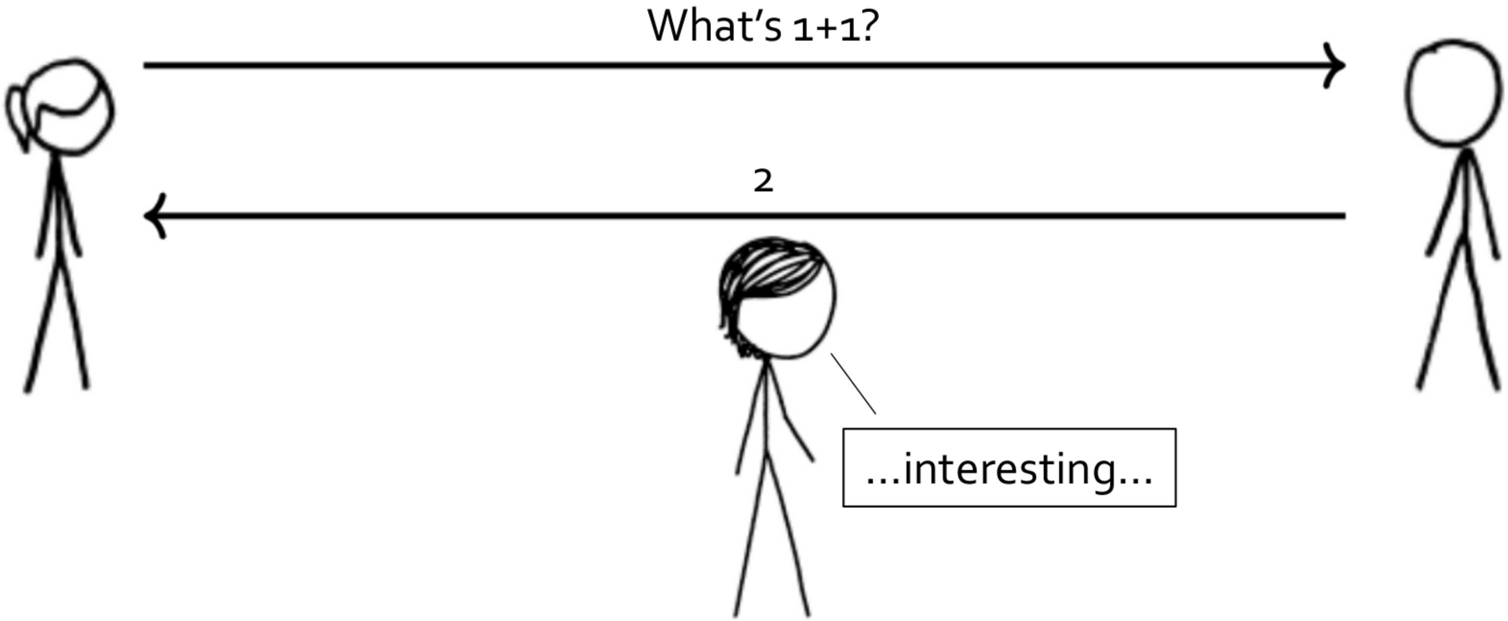
Alice & Bob



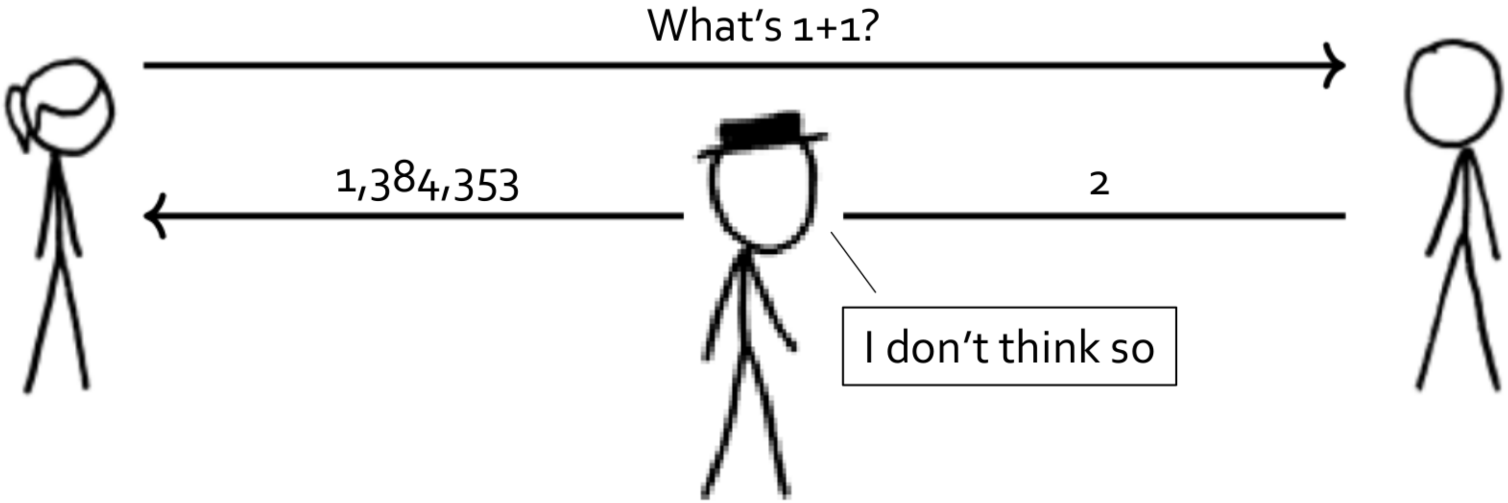
What's 1+1?

2

Eve the Eavesdropper



Malicious Mallory



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

WARNING

I AM NOT A CRYPTOGRAPHER

WARNING

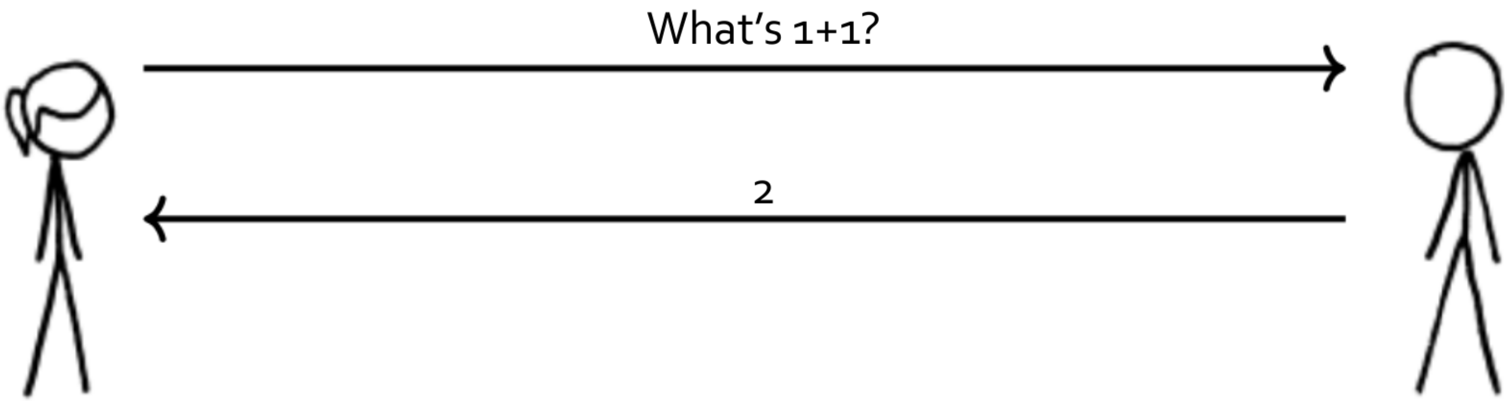
YOU ARE NOT A CRYPTOGRAPHER

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

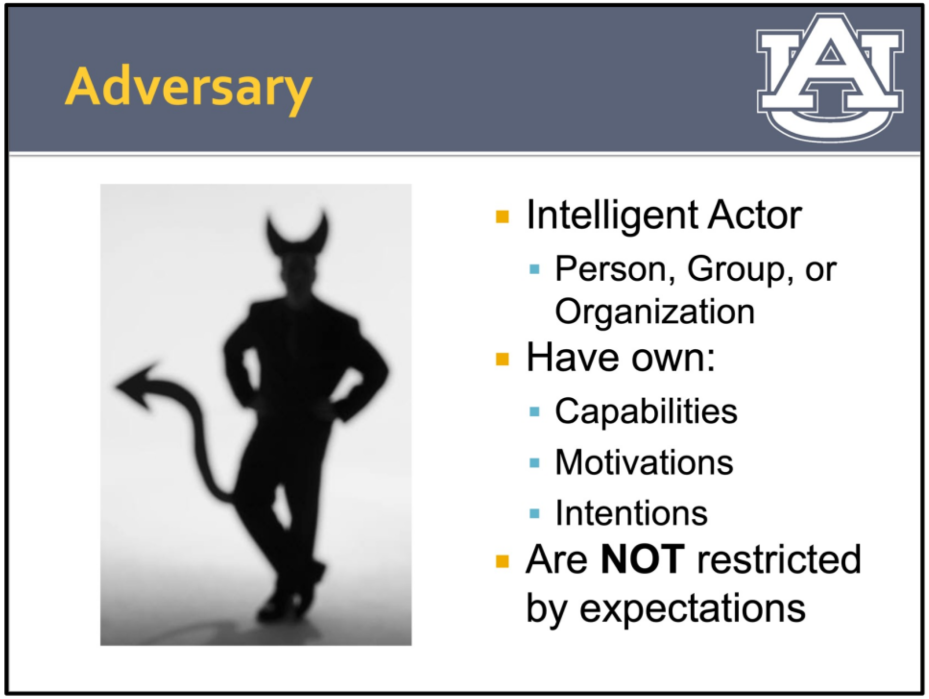
THE SECOND RULE OF CRYPTO
IS DON'T ROLL YOUR OWN CRYPTO
imgflip.com

Building a Secure Channel

- ☹ Confidentiality
- ☹ Message Integrity
- ☹ Sender Authenticity

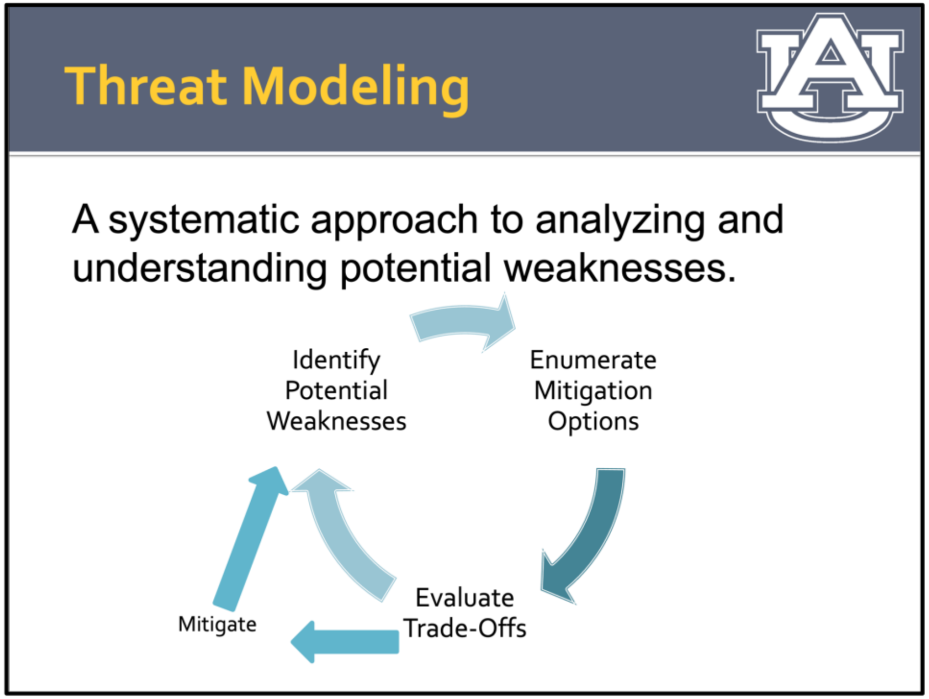


Thinking about Properties



Adversary

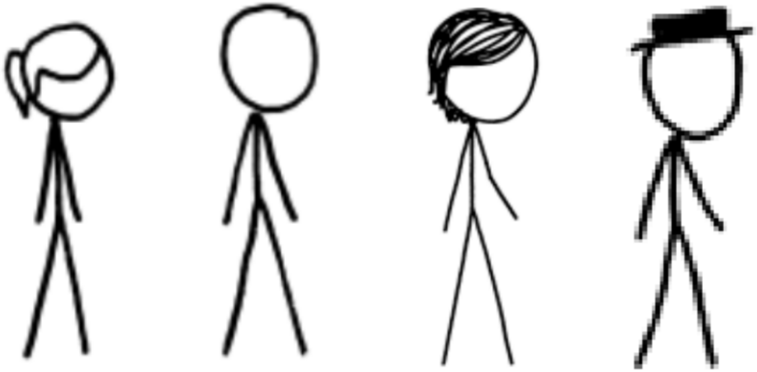
- Intelligent Actor
  - Person, Group, or Organization
- Have own:
  - Capabilities
  - Motivations
  - Intentions
- Are **NOT** restricted by expectations



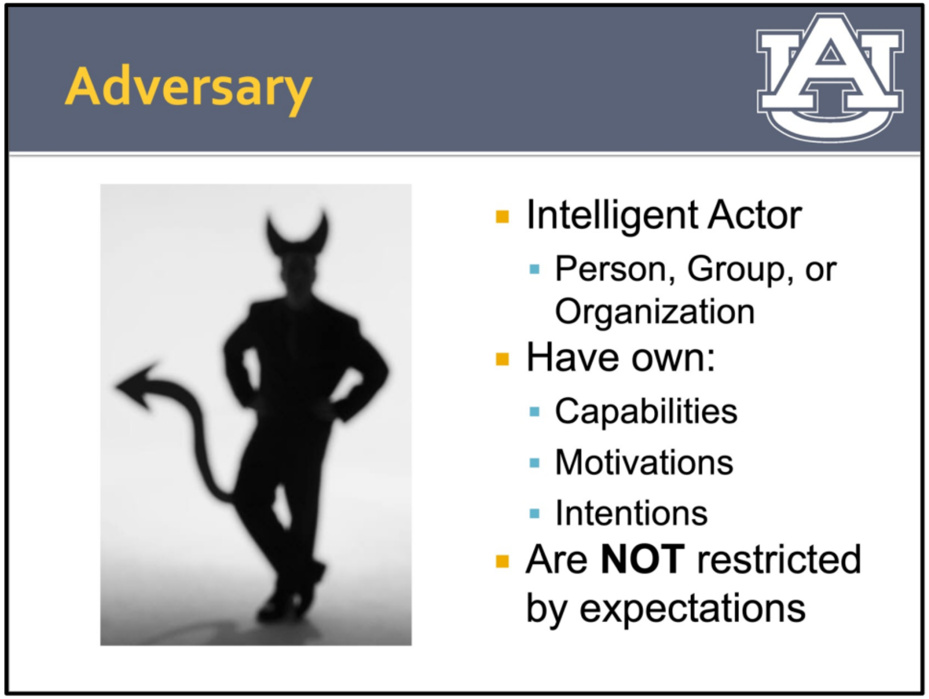
Threat Modeling

A systematic approach to analyzing and understanding potential weaknesses.

For **message integrity**, who should we be worried about?

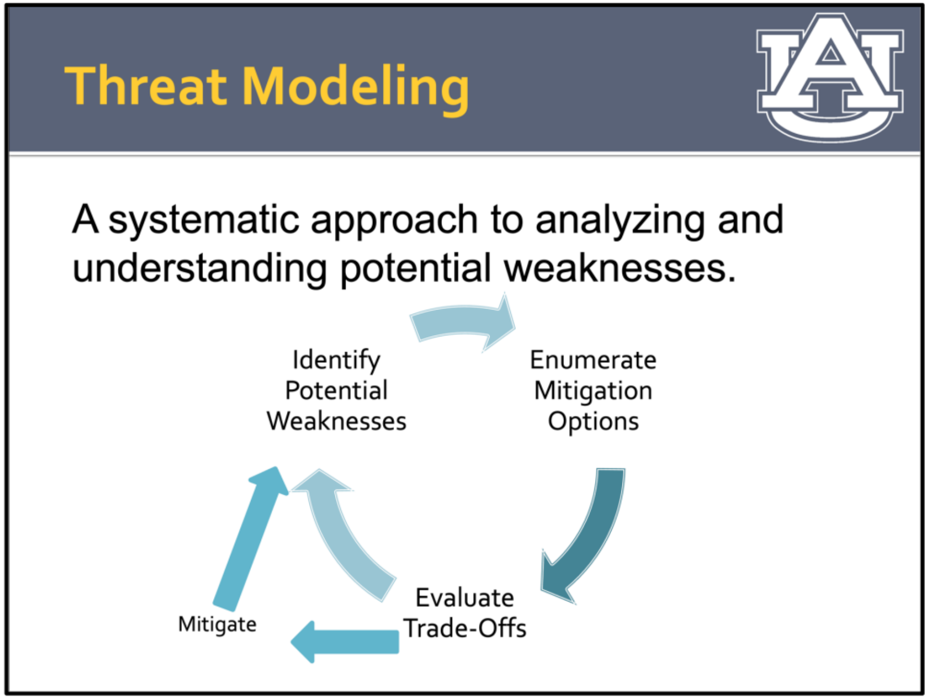


Thinking about Properties



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.

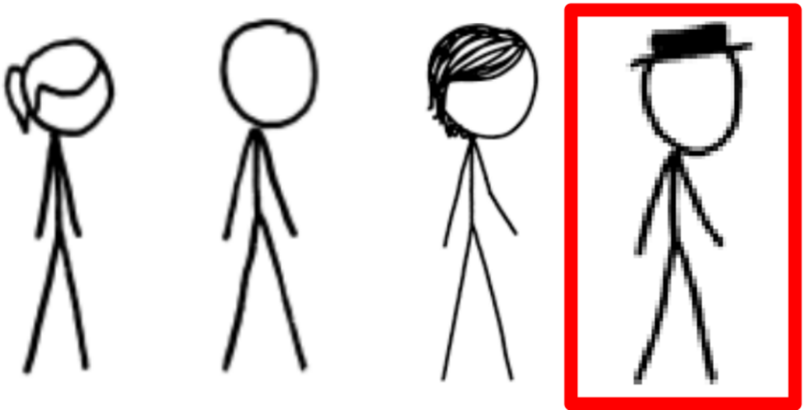
Identify Potential Weaknesses

Enumerate Mitigation Options

Evaluate Trade-Offs

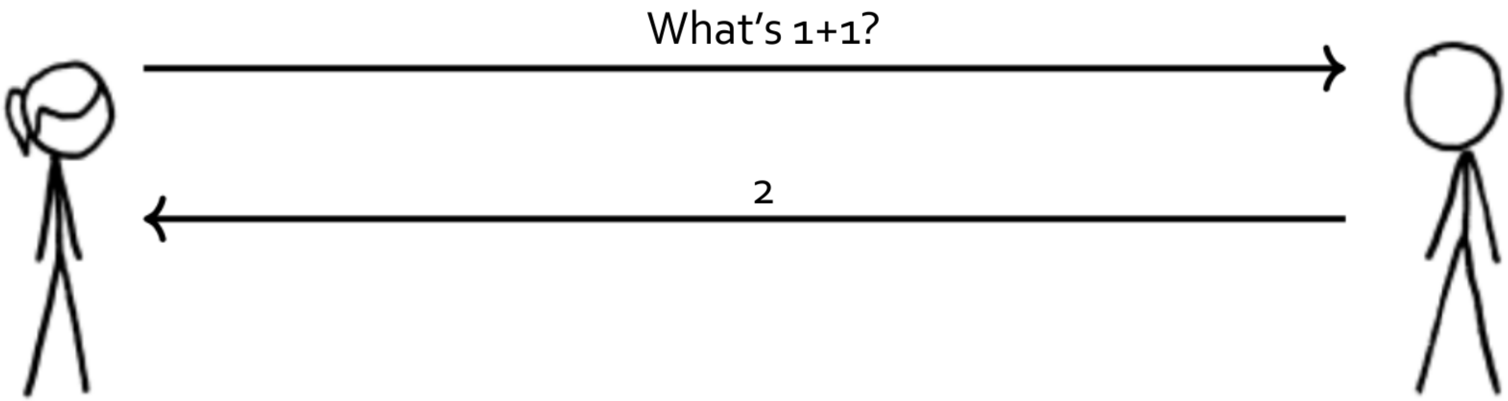
Mitigate

For **message integrity**, who should we be worried about?

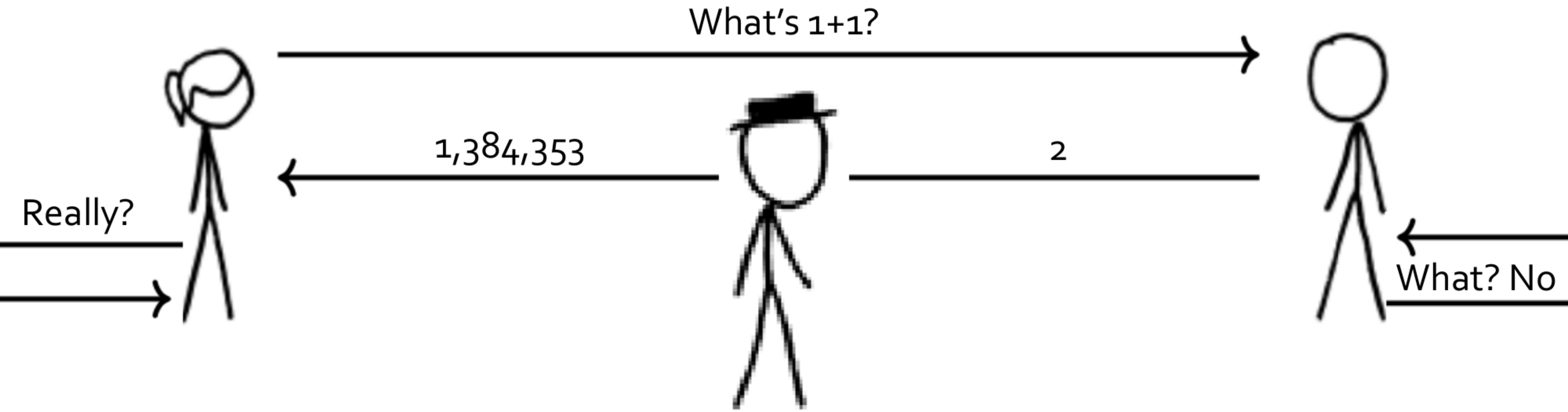


Building a Secure Channel

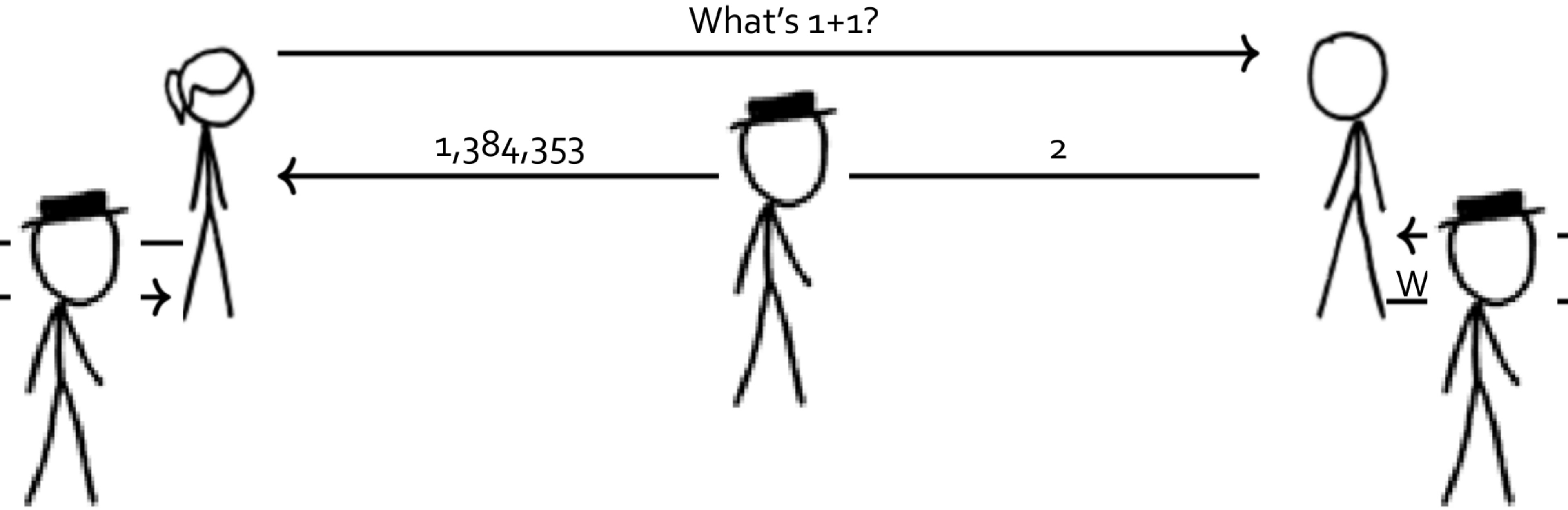
- ☹ Confidentiality
- ☹ Message Integrity
- ☹ Sender Authenticity



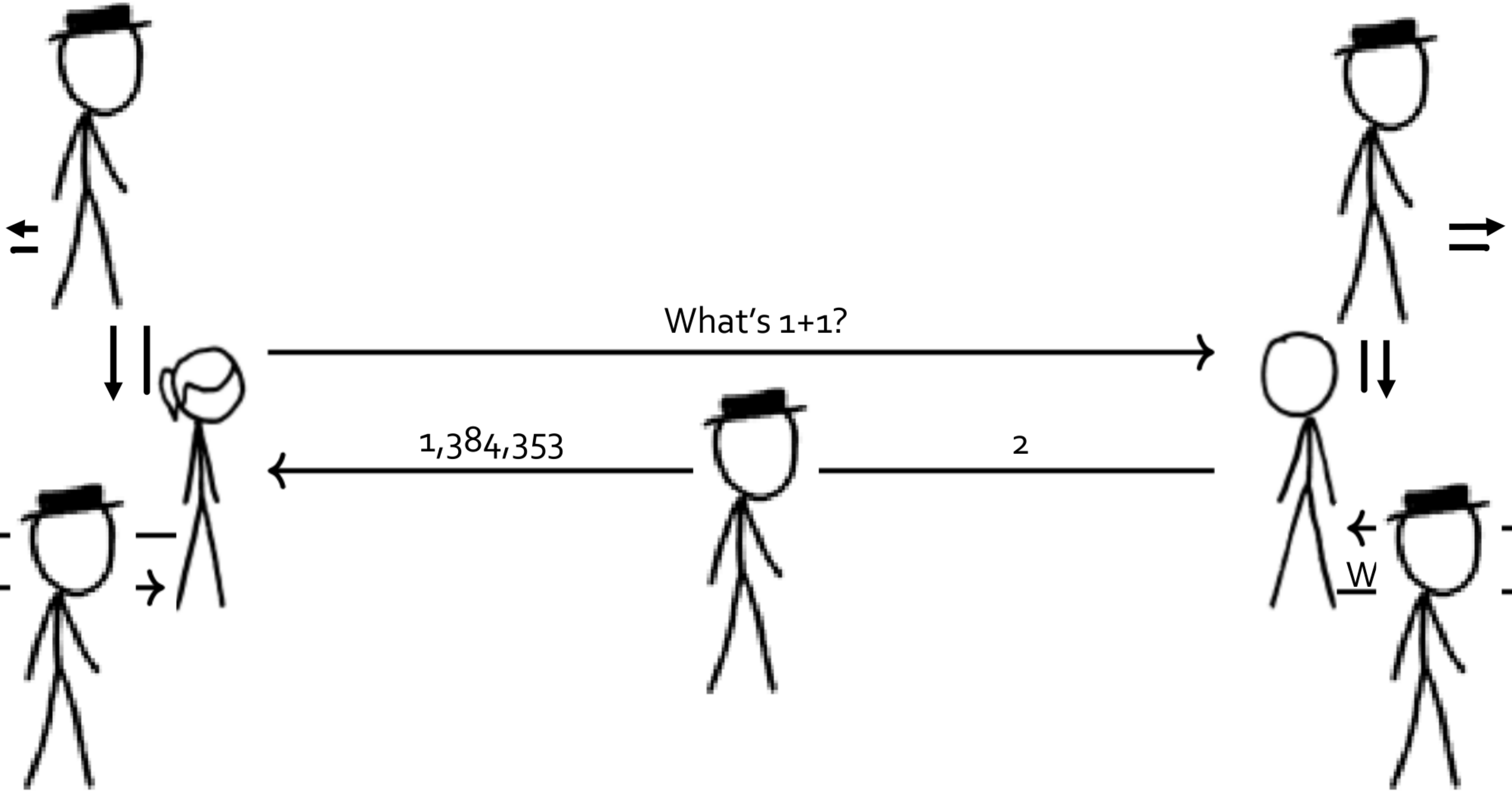
Out-of-Band Validation



Out-of-Band Validation

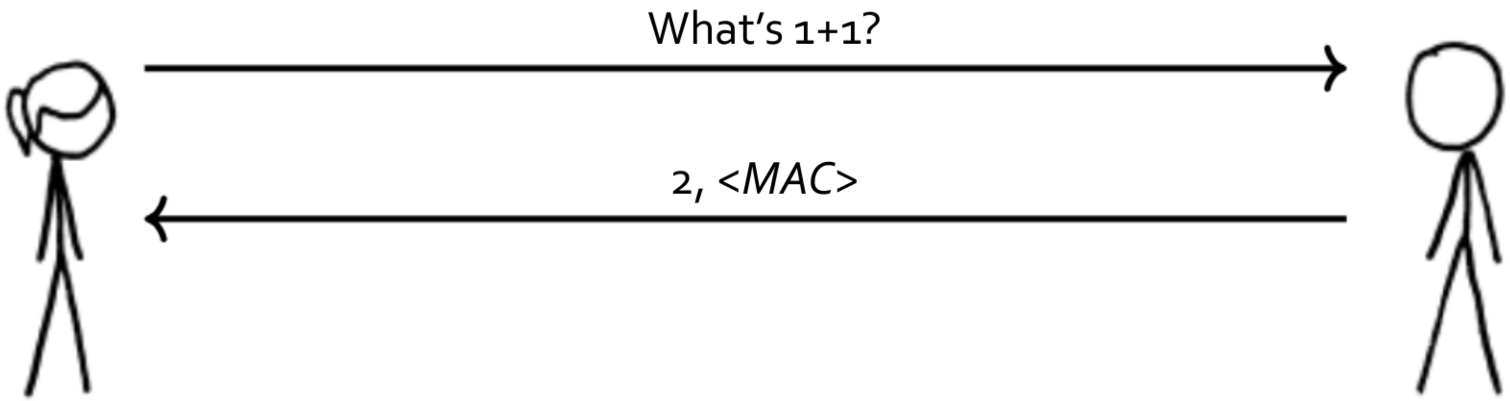


Out-of-Band Validation



Message Authentication Code (MAC)

- Desired attributes of a MAC:
  - Doesn't grow with message length
  - Easy to compute and verify for Alice & Bob
  - Hard for not-Alice/-Bob to create/verify



Pseudorandom Function (PRF)

A **pseudorandom function (PRF)** mimics (but is not) random output regardless of the input.

- Deterministic mapping between in/out
  - (input$_a$ -> output$_a$)
  - (input$_b$ -> output$_b$)
- Output always “looks” random
- If input is unknown, infeasible to recover from output

Hash Function

$$H(x) = y$$

- **Function [H]**
  - 100% public and deterministic
- **Input [x]**
  - Arbitrary length data
- **Output [y]**
  - Fixed-length “digest”

Cryptographic Hash Function

- Collision Resistance
  - Hard to find x_1 and x_2 such that $H(x_1) == H(x_2)$
- Preimage Resistance
  - Given H(x), hard to find x
- Second Preimage Resistance
  - Given x_1, hard to find x_2 such that $H(x_1) == H(x_2)$
- Change Propagation
  - Small input changes make big output changes

Common Hash Functions

	Construction	Year
~~MD5~~	~~Merkle–Damgård~~	~~1992~~
SHA1	Merkle–Damgård	1995
SHA2 (family)	Merkle–Damgård	2001
SHA3/SHAKE (family)	Sponge	2015

MD5

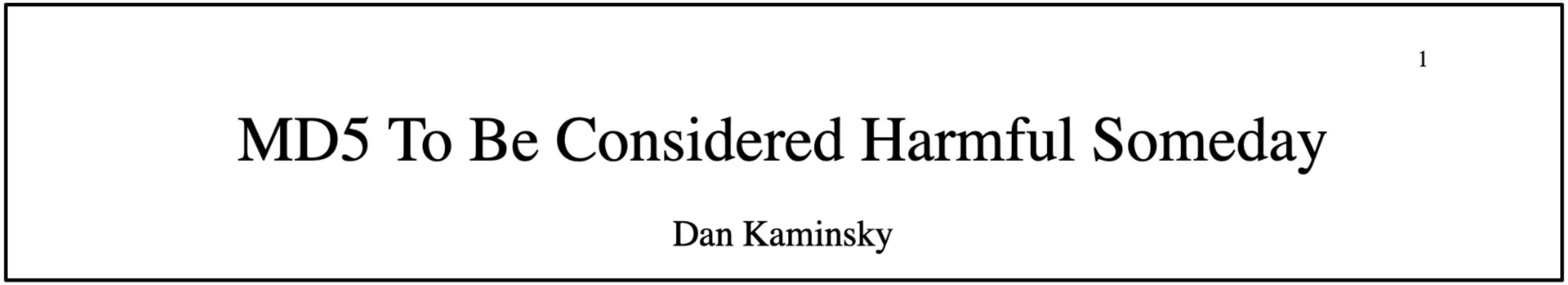
1992 – 2004

Trivial effort to collide

Known use by attackers

NEVER USE…EVER

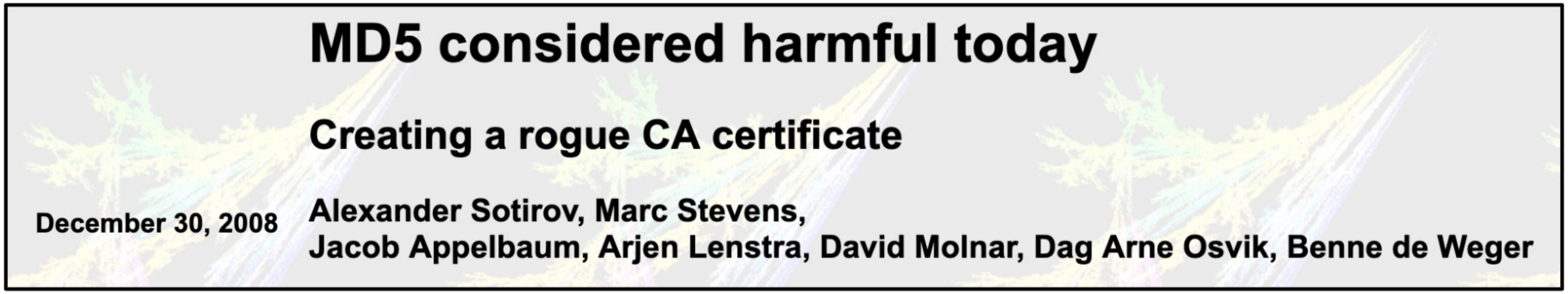
MD5 Collisions



1

MD5 To Be Considered Harmful Someday

Dan Kaminsky



MD5 considered harmful today

Creating a rogue CA certificate

December 30, 2008

Alexander Sotirov, Marc Stevens,
Jacob Appelbaum, Arjen Lenstra, David Molnar, Dag Arne Osvik, Benne de Weger

Common Hash Functions

	Construction	Year
~~MD5~~	~~Merkle–Damgård~~	~~1992~~
~~SHA1~~	~~Merkle–Damgård~~	~~1995~~
SHA2 (family)	Merkle–Damgård	2001
SHA3/SHAKE (family)	Sponge	2015

MD5

1992 – 2004

Trivial effort to collide

Known use by attackers

NEVER USE…EVER

SHA1

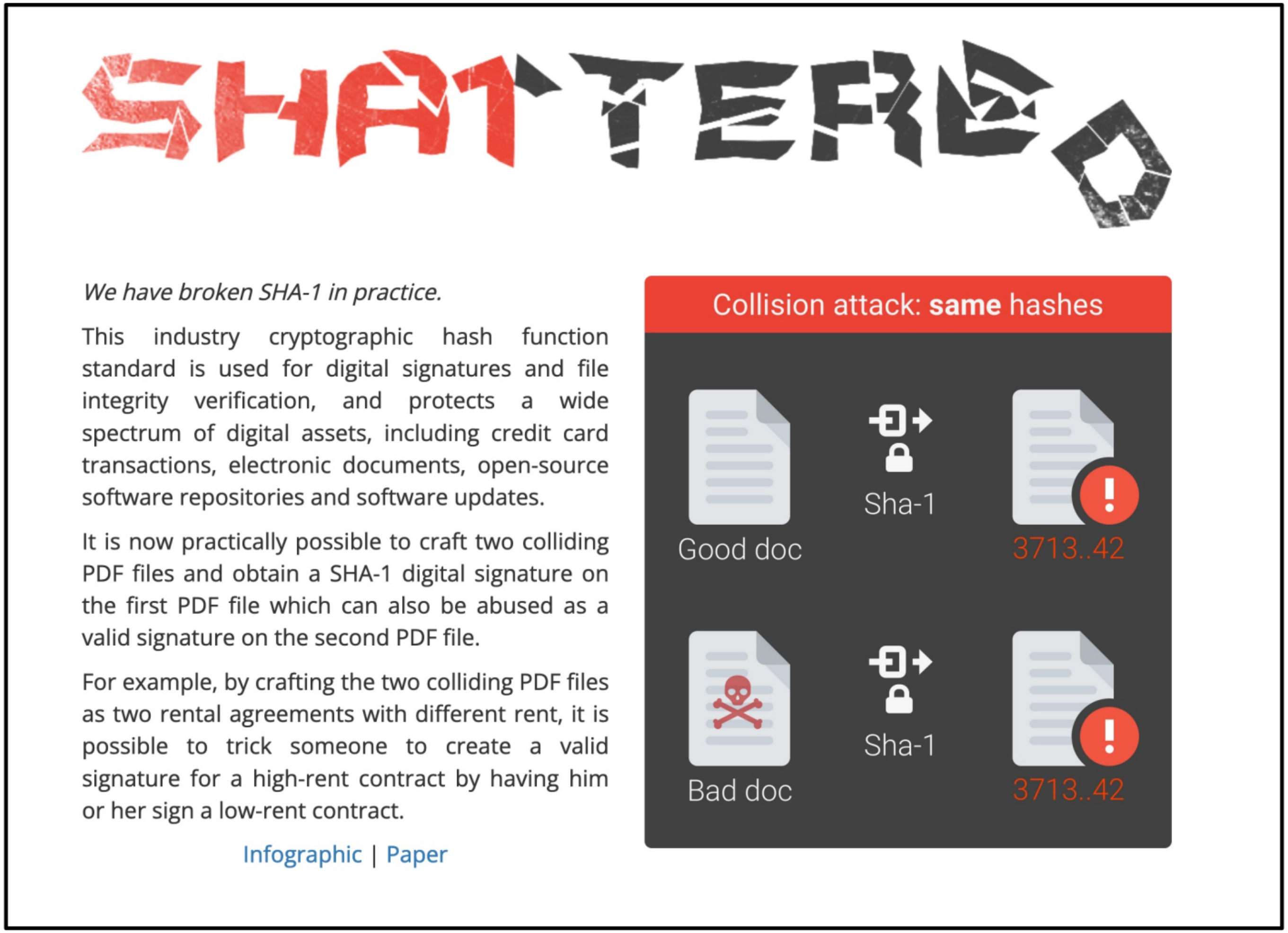
1995 – 2017

Can collide with major effort

Do not use in new systems

Start moving away from

SHA1 Collision



SHATTERED

We have broken SHA-1 in practice.

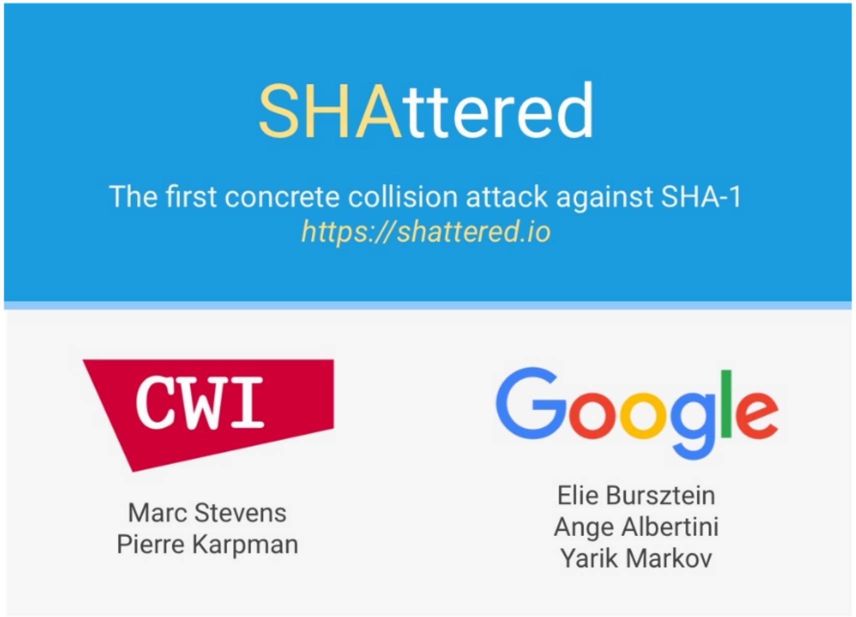
This industry cryptographic hash function standard is used for digital signatures and file integrity verification, and protects a wide spectrum of digital assets, including credit card transactions, electronic documents, open-source software repositories and software updates.

It is now practically possible to craft two colliding PDF files and obtain a SHA-1 digital signature on the first PDF file which can also be abused as a valid signature on the second PDF file.

For example, by crafting the two colliding PDF files as two rental agreements with different rent, it is possible to trick someone to create a valid signature for a high-rent contract by having him or her sign a low-rent contract.

Infographic | Paper

SHA1 Collision



SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

CWI

Marc Stevens
Pierre Karpman

Google

Elie Bursztein
Ange Albertini
Yarik Markov

SHAttered

The first concrete collision attack against SHA-1
https://shattered.io

CWI

Marc Stevens
Pierre Karpman

Google

Elie Bursztein
Ange Albertini
Yarik Markov

Common Hash Functions

	Construction	Year
~~MD5~~	~~Merkle–Damgård~~	~~1992~~
~~SHA1~~	~~Merkle–Damgård~~	~~1995~~
SHA2 (family)	Merkle–Damgård	2001
SHA3/SHAKE (family)	Sponge	2015

MD5

1992 – 2004

Trivial effort to collide

Known use by attackers

NEVER USE…EVER

SHA1

1995 – 2017

Can collide with major effort

Do not use in new systems

Start moving away from

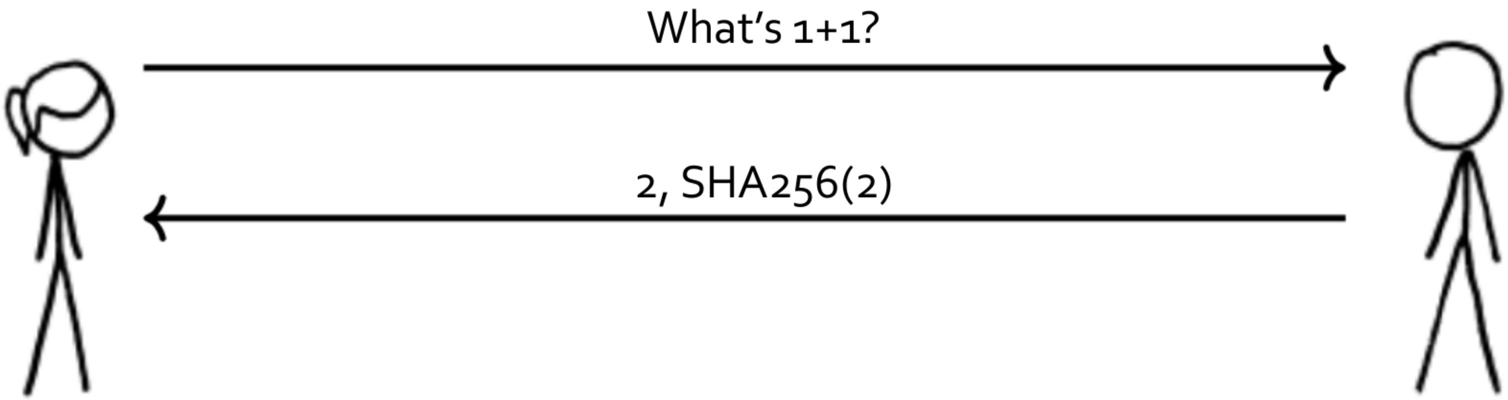
SHA2 Family

- Not perfect but not completely broken
- Comes in a variety of sizes
  - 224, 256, 384, and 512 bits
  - SHA-256 == 256-bit digest

- SHA-256 is OK and widely used
- SHA-384 is approved for CNSA Suite
- SHA3 is OK but relatively low-usage

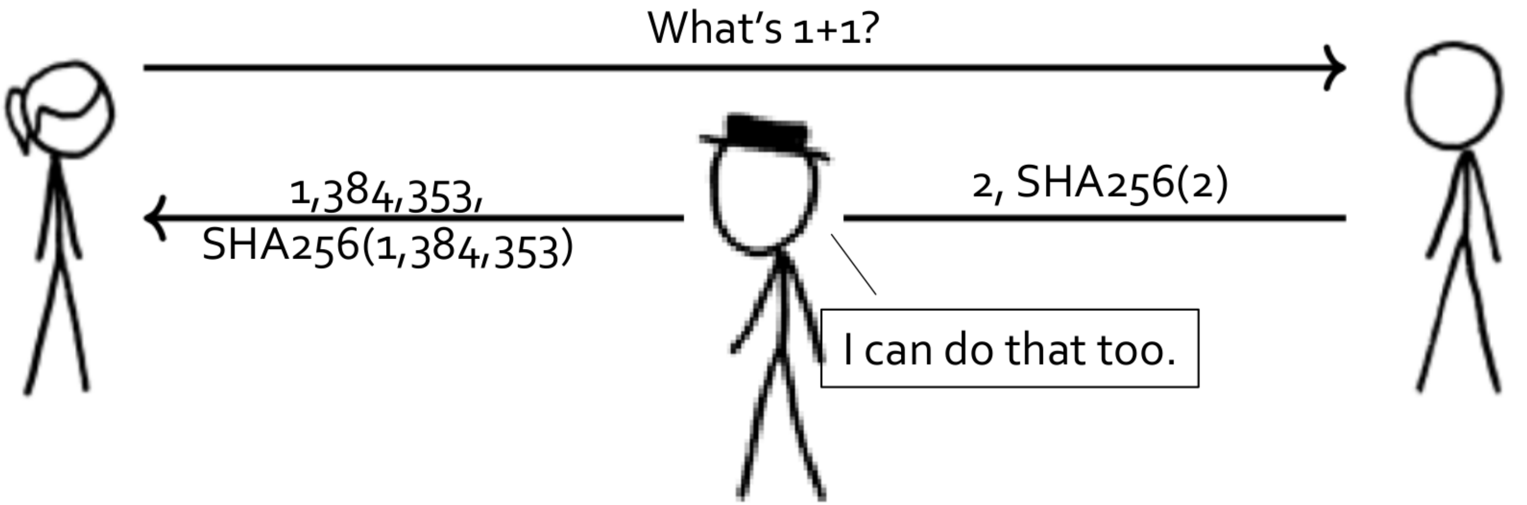
Attempt #1: Hash Function

- Use SHA256 as MAC?
  - Doesn’t grow with message length
  - Easy to compute and verify for Alice & Bob
  - ~~Hard for not-Alice/-Bob to create~~



Attempt #1: Hash Function

- Use SHA256 as MAC?
  - Doesn't grow with message length
  - Easy to compute and verify for Alice & Bob
  - ~~Hard for not-Alice/-Bob to create~~



Safely Using a Hash Function

Hash functions are still very, very useful.

H(x) = y

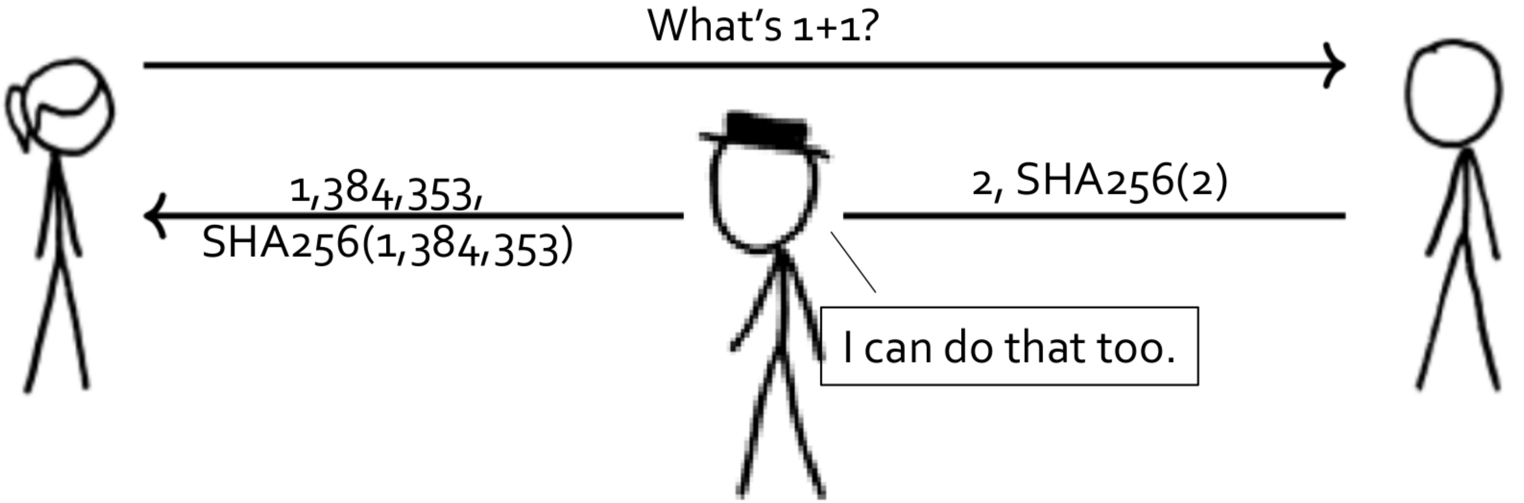
- **Function [H]**
  - 100% public and deterministic
- **Input [x]**
  - Arbitrary length data
- **Output [y]**
  - Fixed-length "digest"

- Collision Resistance
  - Hard to find x_1 and x_2 such that $H(x_1) == H(x_2)$
- Preimage Resistance
  - Given H(x), hard to find x
- Second Preimage Resistance
  - Given x_1, hard to find x_2 such that $H(x_1) == H(x_2)$
- Change Propagation
  - Small input changes make big output changes

Where/When can you **safely** use a raw hash?

Attempt #1: Hash Function

- Use SHA256 as MAC?
  - Easy to compute and verify for Alice & Bob
  - Doesn’t grow with message length
  - ~~Hard for not-Alice/-Bob to create~~



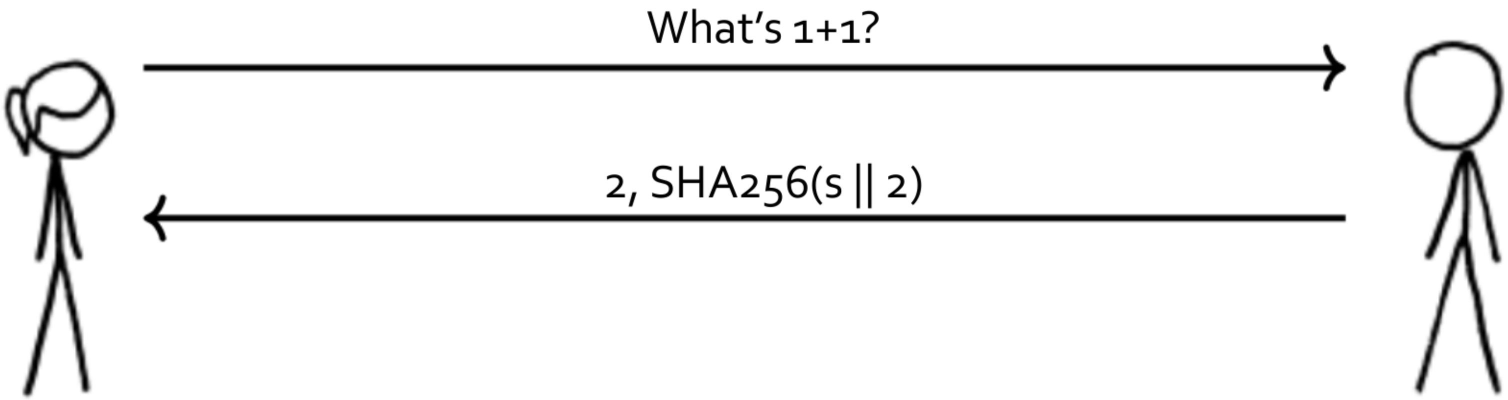
Attempt #2: Hash w/ Secret

$$H(s \mid\mid x) = y$$

- **Function [H]**
  - 100% public and deterministic
- **Secret [s]**
  - Is only known to 1P actors
- **Input [x]**
  - Arbitrary length data
- **Output [y]**
  - Fixed-length “digest”

Attempt #2: Hash w/ Secret

- Use SHA256 w/ secret as MAC?
  - Easy to compute and verify for Alice & Bob
  - Doesn't grow with message length
  - ~~Hard for not-Alice/-Bob to create~~



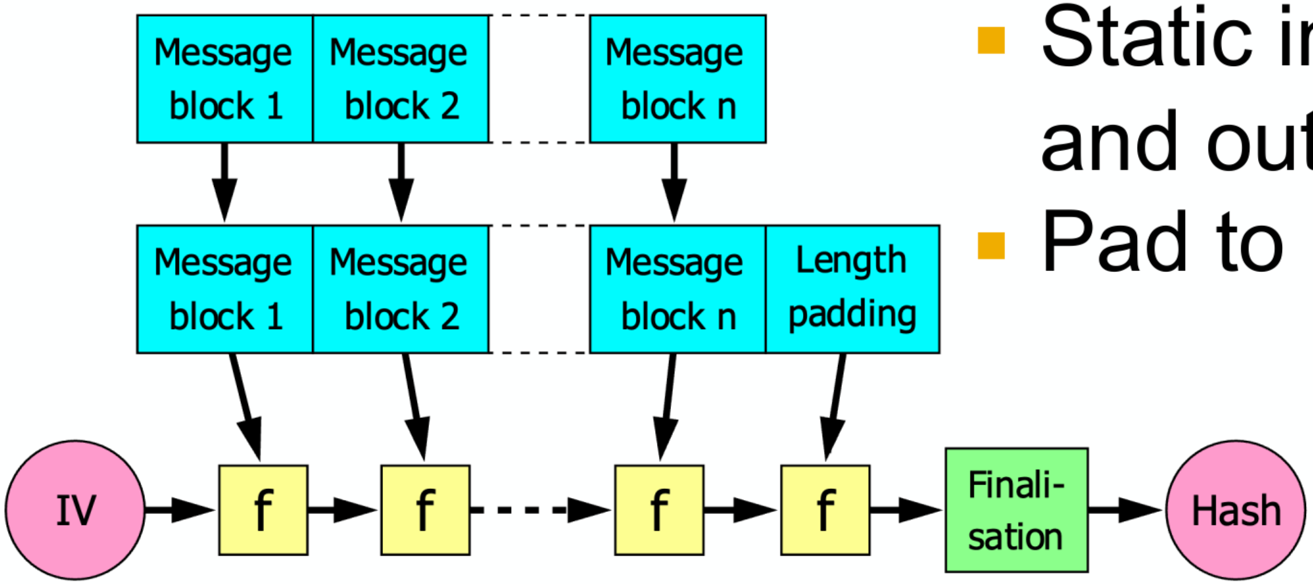
What's 1+1?

2, SHA256(s || 2)

Merkle–Damgård

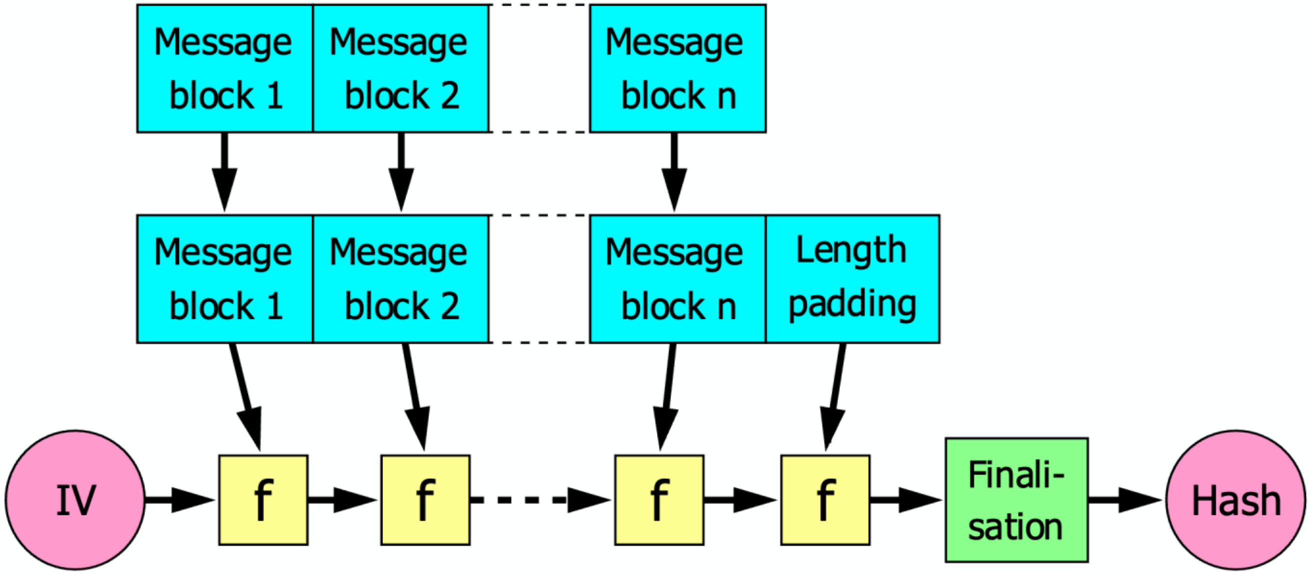
Many hash functions use **Merkle–Damgård construction** with a hash-specific compression function.

- Break message into constant-size blocks
- Static internal-state and output size
- Pad to block-length

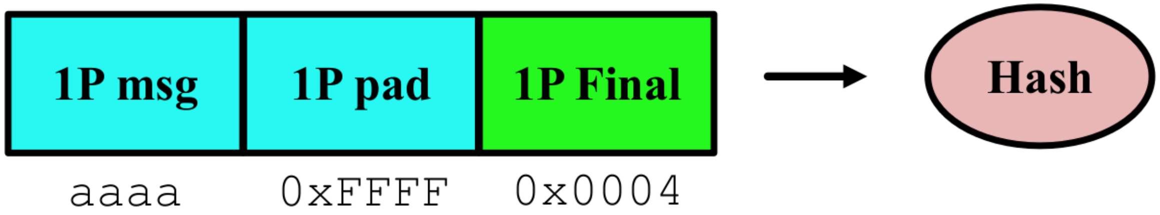


Length Extension Attacks

An attacker uses a known-hash for a known-length but unknown-content message to create hash for a partially-controlled message prefixed by the unknown message.



Attack Example (simplified)



1P msg

1P pad

1P Final

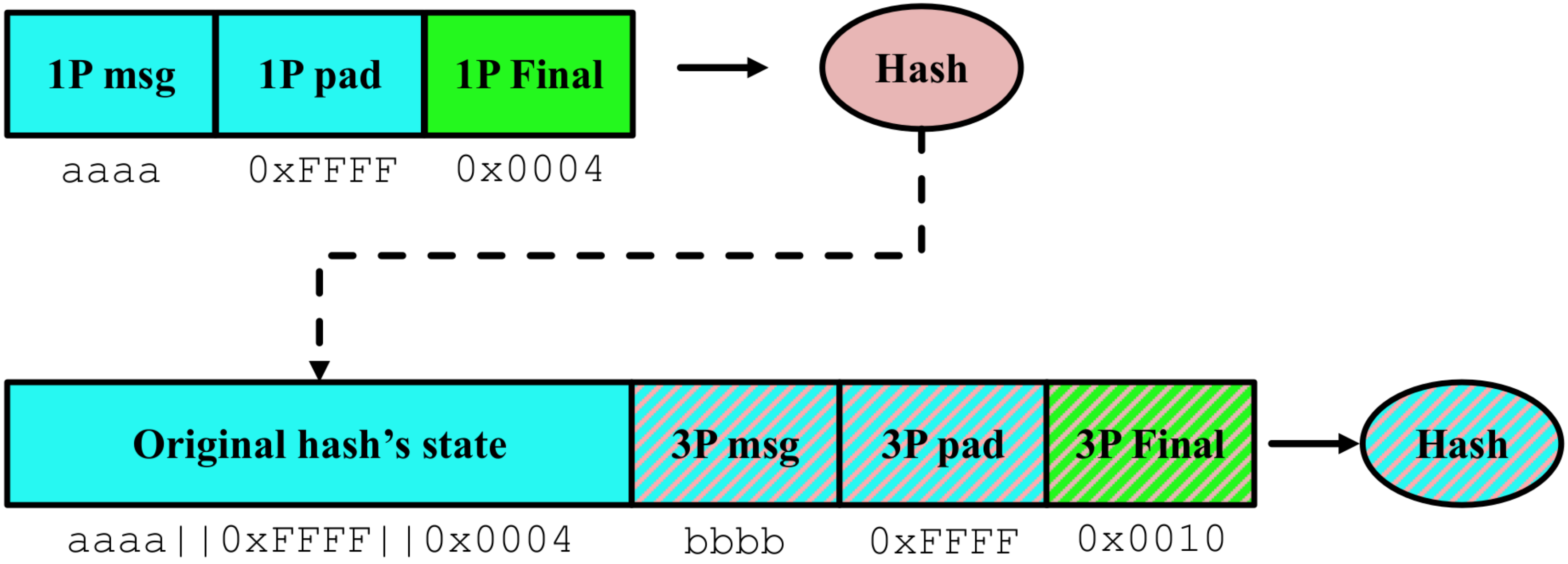
Hash

aaaa

0xFFFF

0x0004

Attack Example (simplified)



Attack Example (simplified)

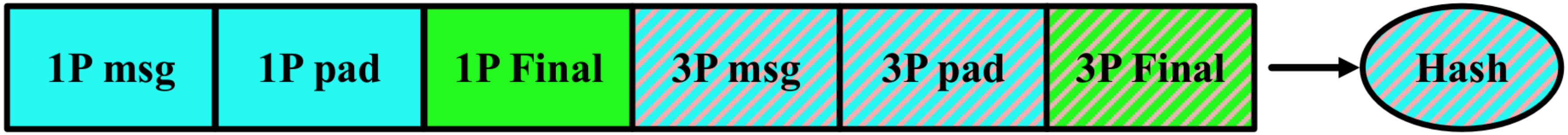
- 1P message: “Let’s go to the mall”
- 3P message: “next week”

- Message according to hash function:

```
‘Let’s go to the mall’
   + padding
      + length
         + ‘next week’
```

Attack Example (URL)

Source: https://en.wikipedia.org/wiki/Length_extension_attack



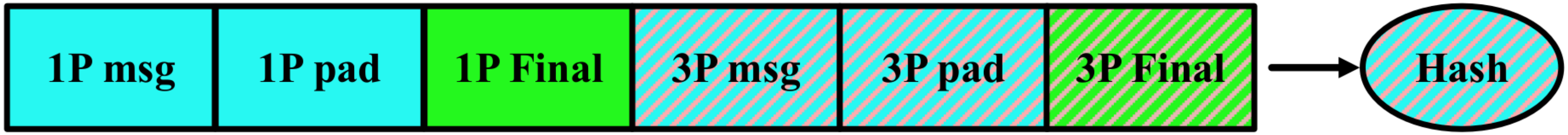
```
order.com/count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo
```

- Change to different type of waffle
  - `waffle=liege`

```
order.com/count=10&lat=37.351&user_id=1&long=-
119.827&waffle=eggo\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x02\x28&waffle=liege
```

Attack Example (URL)

Source: https://en.wikipedia.org/wiki/Length_extension_attack



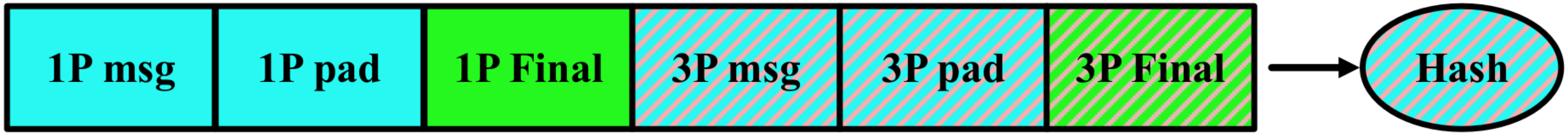
`order.com/count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo`

- Change to different type of waffle
  - `waffle=liege`

order.com/count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x02\x28&waffle=liege

Attack Example (URL)

Source: https://en.wikipedia.org/wiki/Length_extension_attack



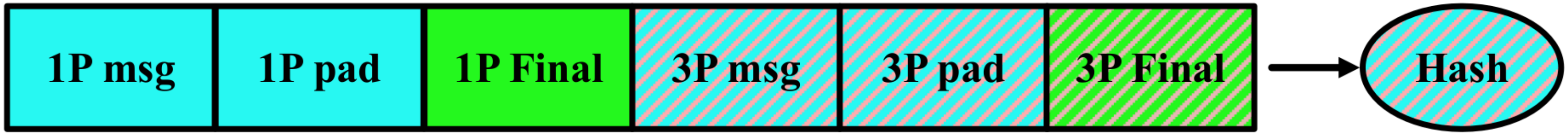
order.com/count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo

- Change to different type of waffle
  - waffle=liege

order.com/count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo**\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00**\x00\x00\x02\x28&waffle=liege

Attack Example (URL)

Source: https://en.wikipedia.org/wiki/Length_extension_attack



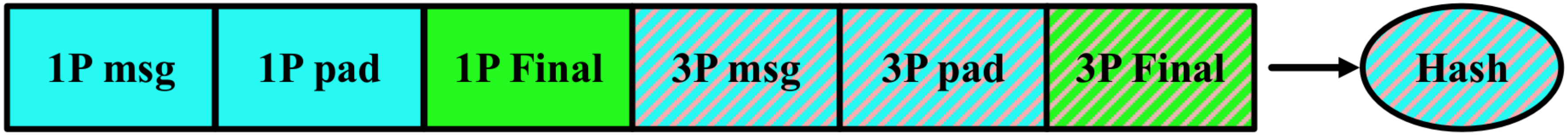
order.com/count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo

- Change to different type of waffle
  - waffle=liege

order.com/count=10&lat=37.351&user_id=1&long=-
119.827&waffle=eggo\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00**\x00\x00\x02\x28**&waffle=liege

Attack Example (URL)

Source: https://en.wikipedia.org/wiki/Length_extension_attack


1P msg | 1P pad | 1P Final | 3P msg | 3P pad | 3P Final → Hash

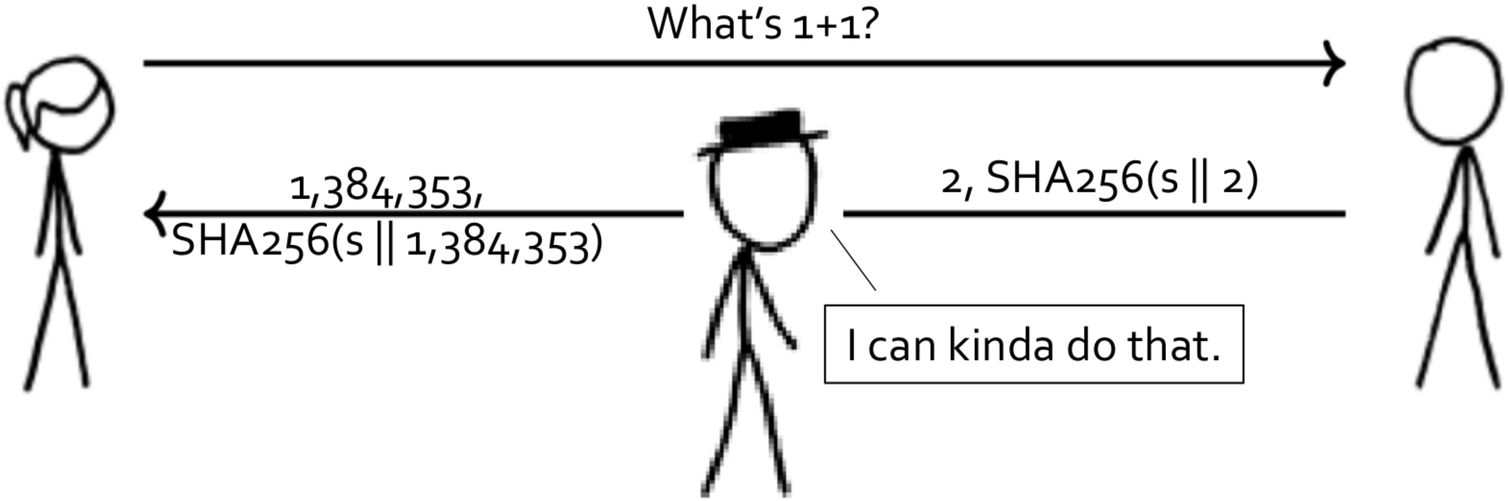
```
order.com/count=10&lat=37.351&user_id=1&long=-119.827&waffle=eggo
```

- Change to different type of waffle
  - `waffle=liege`

```
order.com/count=10&lat=37.351&user_id=1&long=-
119.827&waffle=eggo\x80\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\
x00\x00\x00\x00\x00\x00\x00\x00\x02\x28&waffle=liege
```

Attempt #2: Hash w/ Secret

- Use SHA256 w/ secret as MAC?
  - Easy to compute and verify for Alice & Bob
  - Doesn't grow with message length
  - ~~Hard for not-Alice/-Bob to create~~



Attempt #3: HMAC

HMAC(s, x) = y

- **Function [HMAC]**
  - “Hash-Based Message Authentication Code”
  - Specific usage of hash functions
- **Secret [s]**
  - Is only known to 1P actors
- **Input [x]**
  - Arbitrary length data
- **Output [y]**
  - Fixed-length “digest”

Turning a Hash into an HMAC

- Any hash function can be turned into an HMAC using a simple construction

$$\mathrm{HMAC}(K,m) = \mathrm{H}\Big(\big(K' \oplus opad\big) \,\|\, \mathrm{H}\big(\big(K' \oplus ipad\big) \,\|\, m\big)\Big)$$

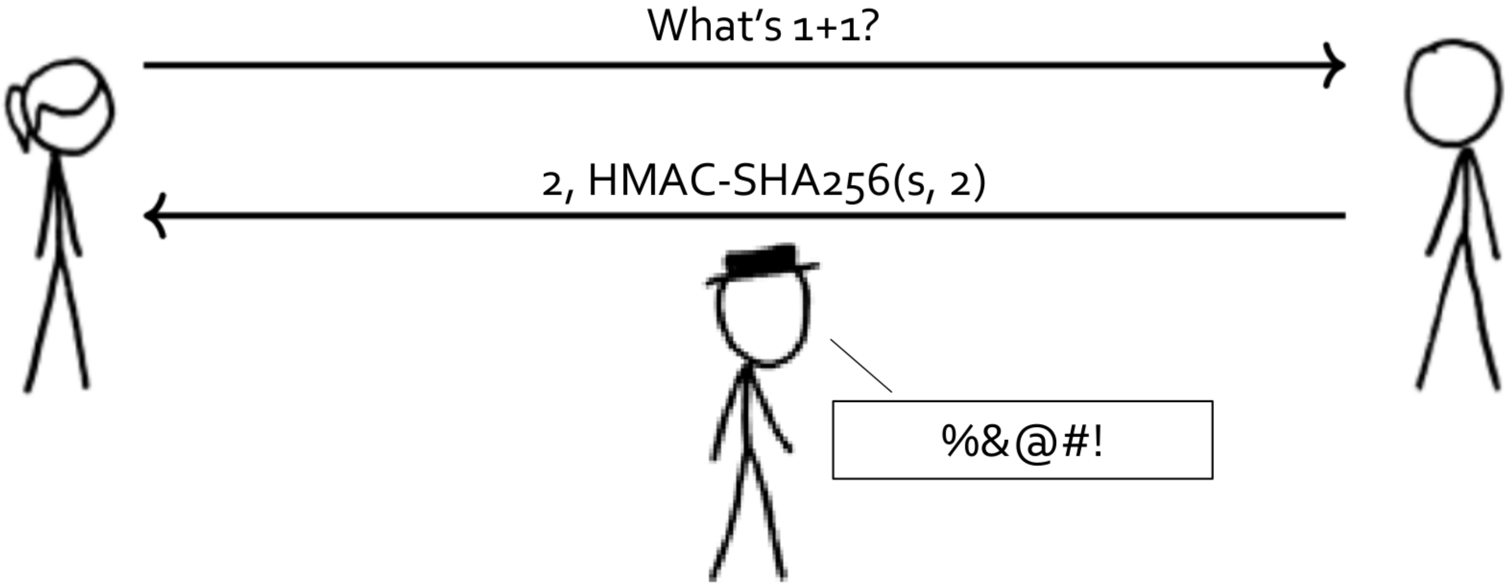
$$K' = \begin{cases} \mathrm{H}(K) & K \text{ is larger than block size} \\ K & \text{otherwise} \end{cases}$$

**opad* and *ipad* are block-sized constants

- HMAC-SHA256 == HMAC using SHA-256

Attempt #3: HMAC

- Use HMAC-SHA256 as MAC?
  - Easy to compute and verify for Alice & Bob
  - Doesn't grow with message length
  - Hard for not-Alice/-Bob to create

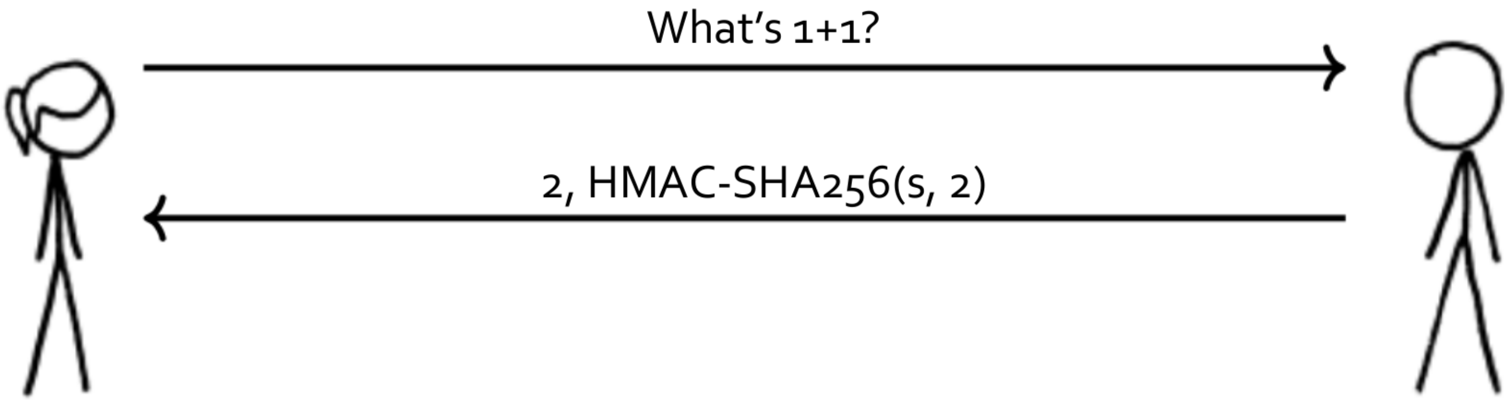


Building a Secure Channel

☹ Confidentiality

☺ Message Integrity

☺ ☹ Sender Authenticity



What's 1+1?

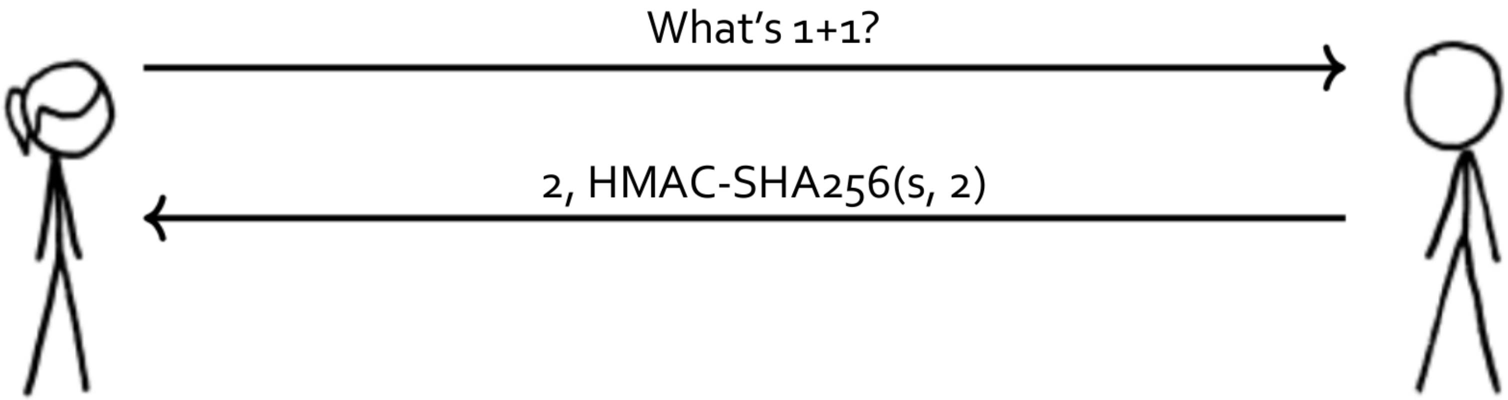
2, HMAC-SHA256(s, 2)

Building a Secure Channel

☹ Confidentiality

☺ Message Integrity

☹ Sender Authenticity ????



What's 1+1?

2, HMAC-SHA256(s, 2)

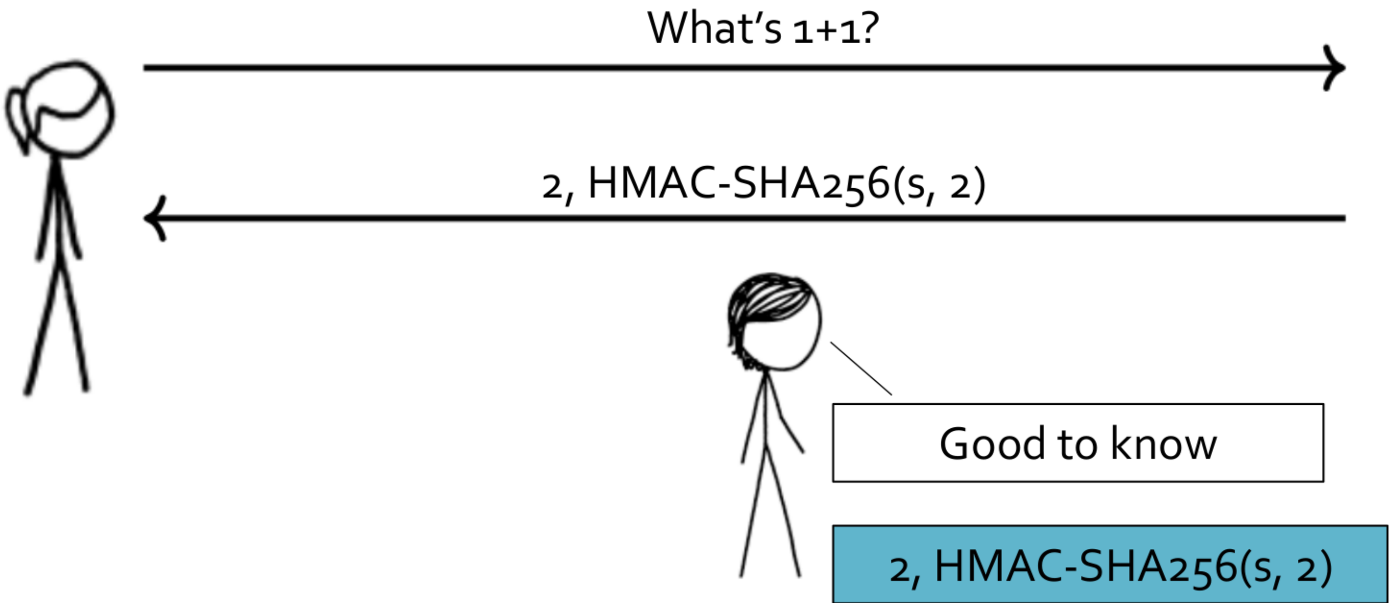
Properties of Secure Channel

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

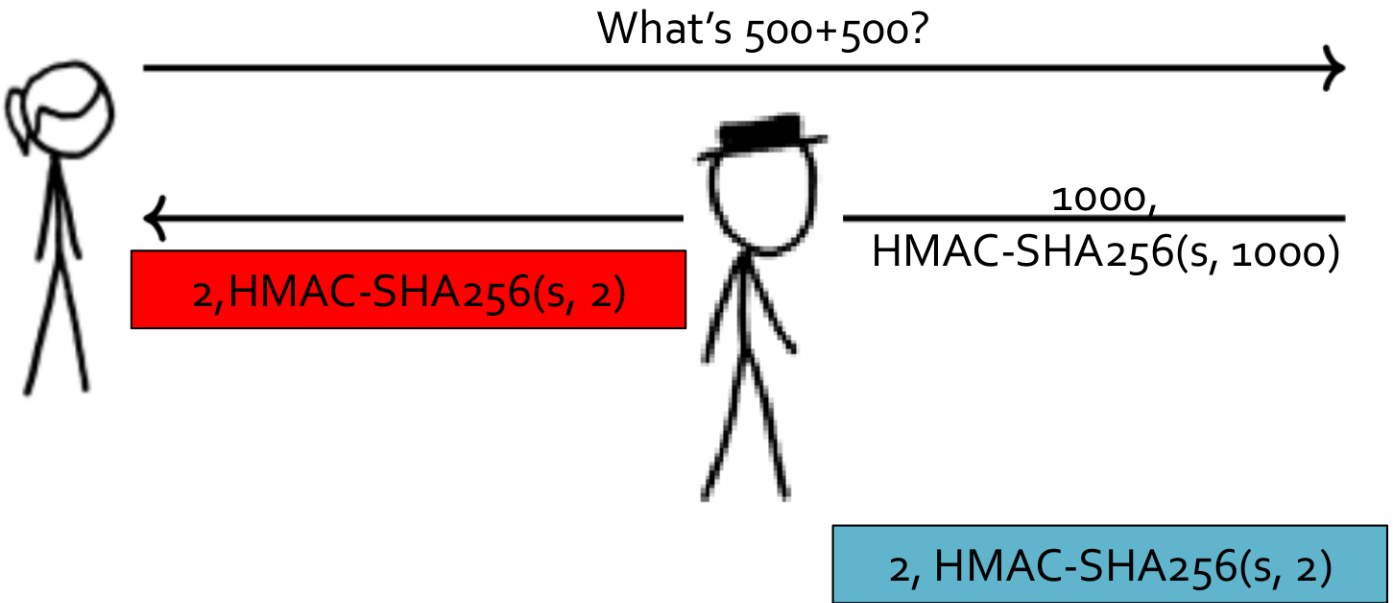
Replay Attacks

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.



What’s 500+500?

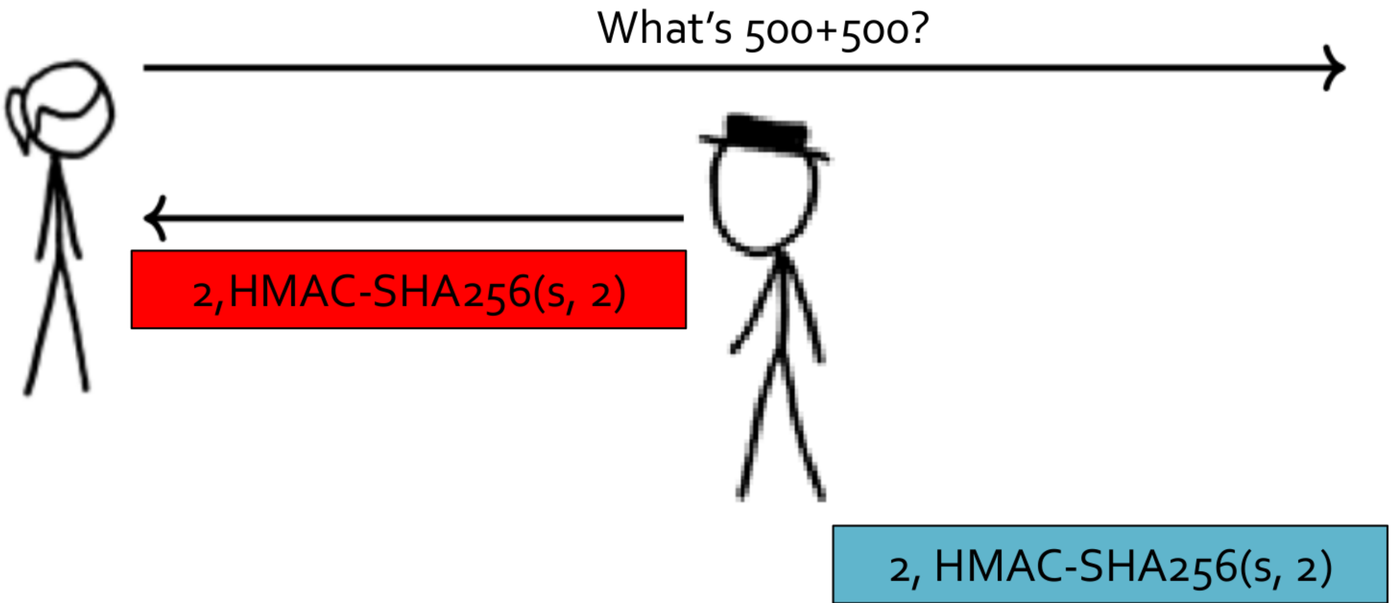
1000, HMAC-SHA256(s, 1000)

2,HMAC-SHA256(s, 2)

2, HMAC-SHA256(s, 2)

Replay Attacks

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



What’s 500+500?

2,HMAC-SHA256(s, 2)

2, HMAC-SHA256(s, 2)

Properties of Secure Channel

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

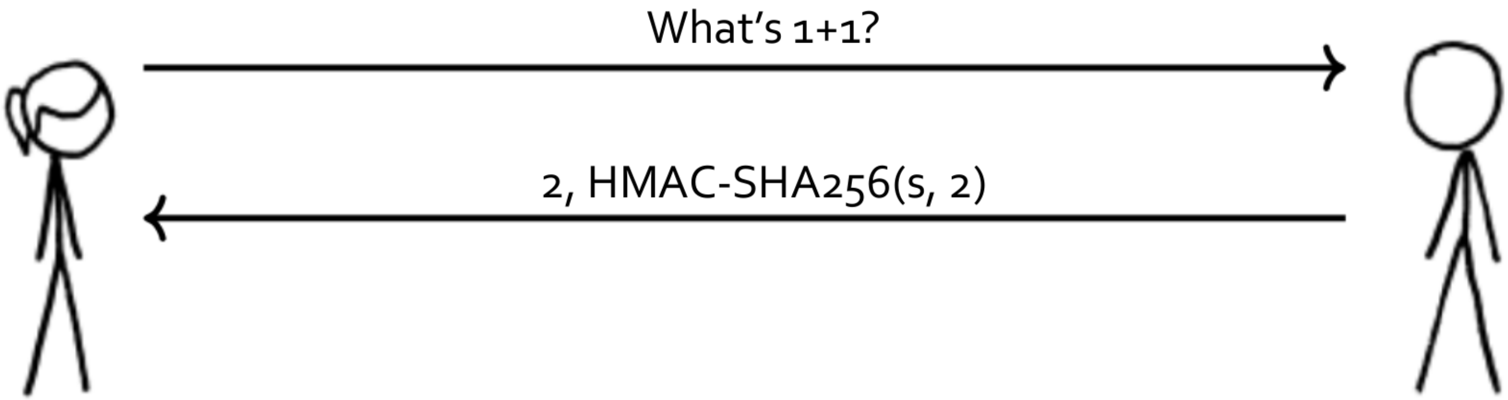
- **Confidentiality**
  - Messages can’t be read by a 3rd party (3P)
- **Message Integrity**
  - Messages can’t be unknowingly modified by 3P
- **Sender Authenticity**
  - Valid messages creatable **only** by a 1P actor

Building a Secure Channel

☹ Confidentiality

☺ Message Integrity

☹ Sender Authenticity



What's 1+1?

2, HMAC-SHA256(s, 2)

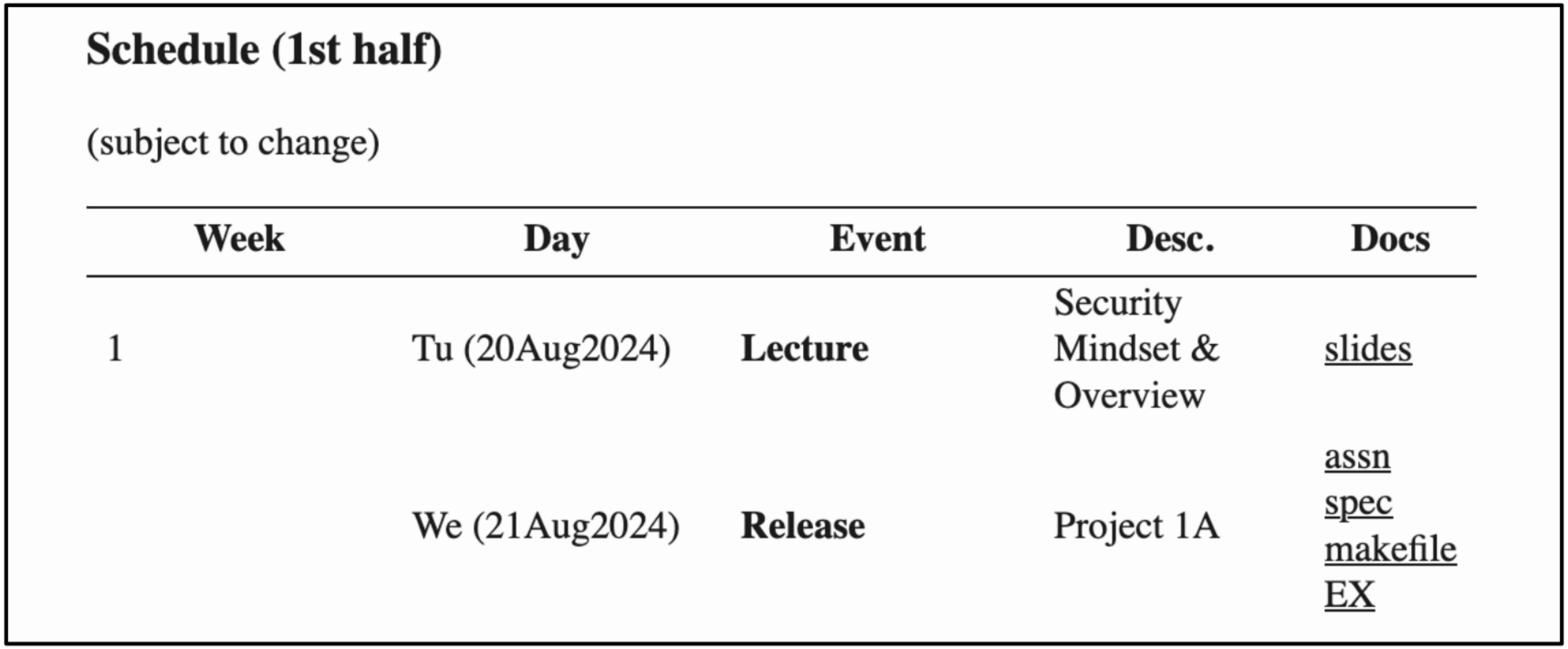
Computer and Network Security

Lecture 03:
Hashing and Integrity

COMP-5370/6370
Fall 2024

Course Notes

- Project 1A is live and due in two weeks

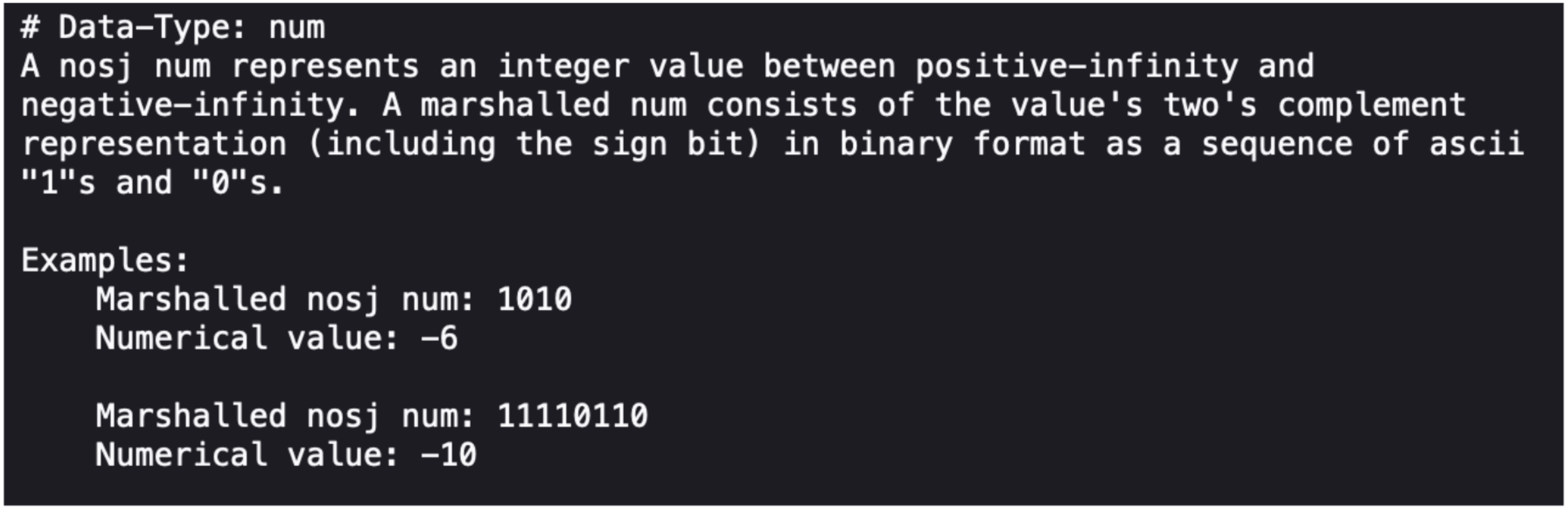


Schedule (1st half)

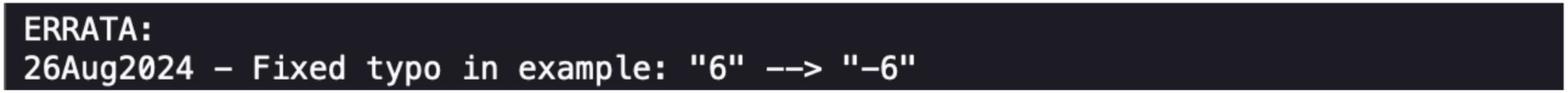
(subject to change)

Week	Day	Event	Desc.	Docs
1	Tu (20Aug2024)	**Lecture**	Security Mindset & Overview	slides
	We (21Aug2024)	**Release**	Project 1A	assn spec makefile EX

Typo Fixed in the Spec



```
# Data-Type: num
A nosj num represents an integer value between positive-infinity and
negative-infinity. A marshalled num consists of the value's two's complement
representation (including the sign bit) in binary format as a sequence of ascii
"1"s and "0"s.

Examples:
    Marshalled nosj num: 1010
    Numerical value: -6

    Marshalled nosj num: 11110110
    Numerical value: -10
```



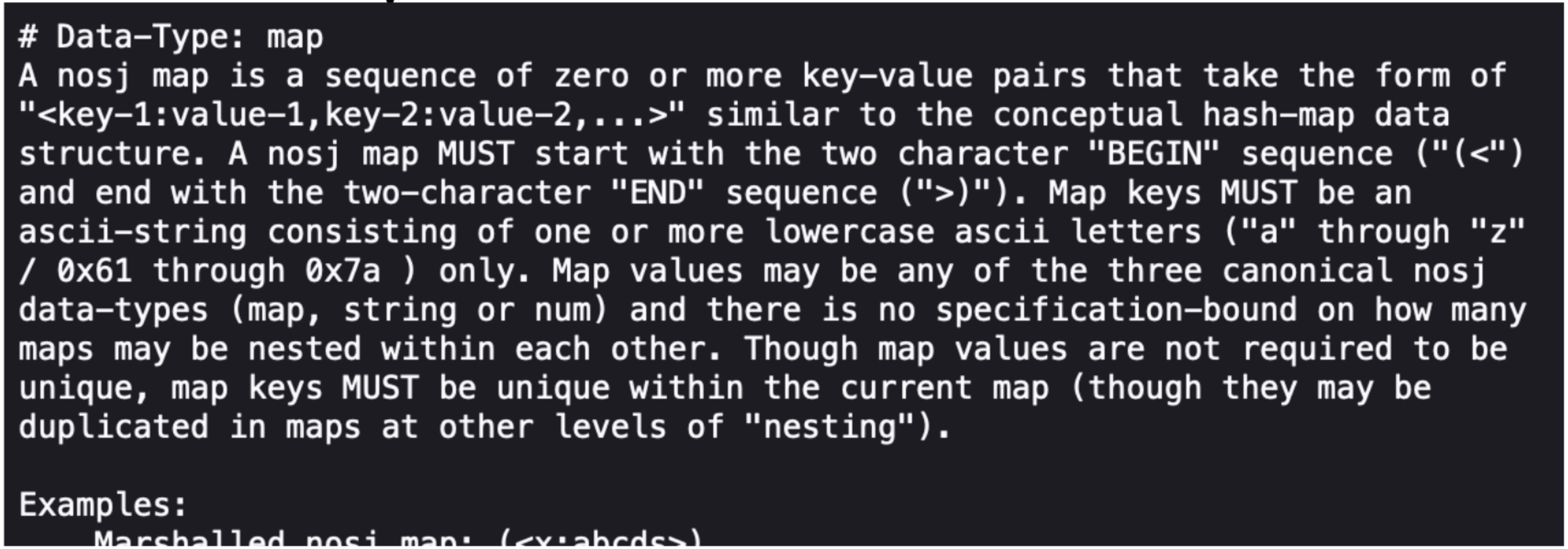
```
ERRATA:
26Aug2024 - Fixed typo in example: "6" --> "-6"
```

Project 1A

Input: `(<abc:defs>)`

Project 1A

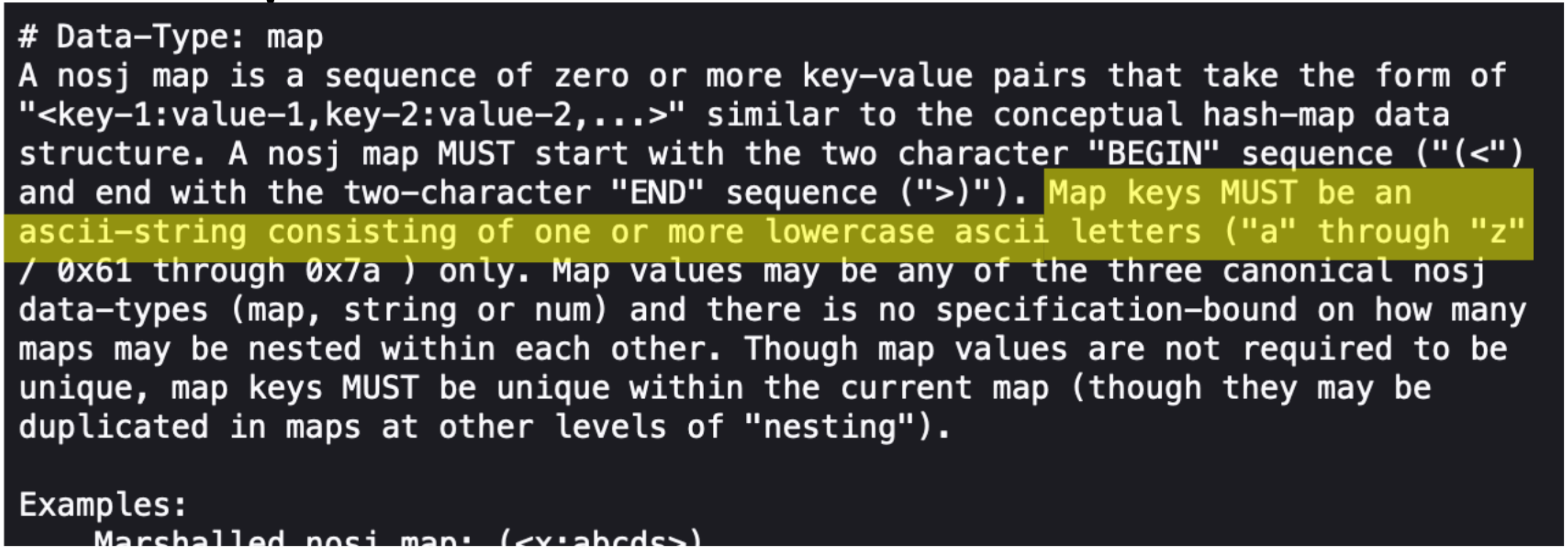
Input: (<abc:defs>)



```
# Data-Type: map
A nosj map is a sequence of zero or more key-value pairs that take the form of
"<key-1:value-1,key-2:value-2,...>" similar to the conceptual hash-map data
structure. A nosj map MUST start with the two character "BEGIN" sequence ("(<")
and end with the two-character "END" sequence (">)"). Map keys MUST be an
ascii-string consisting of one or more lowercase ascii letters ("a" through "z"
/ 0x61 through 0x7a ) only. Map values may be any of the three canonical nosj
data-types (map, string or num) and there is no specification-bound on how many
maps may be nested within each other. Though map values are not required to be
unique, map keys MUST be unique within the current map (though they may be
duplicated in maps at other levels of "nesting").

Examples:
    Marshalled nosj map: (<x:abcds>)
```

Project 1A

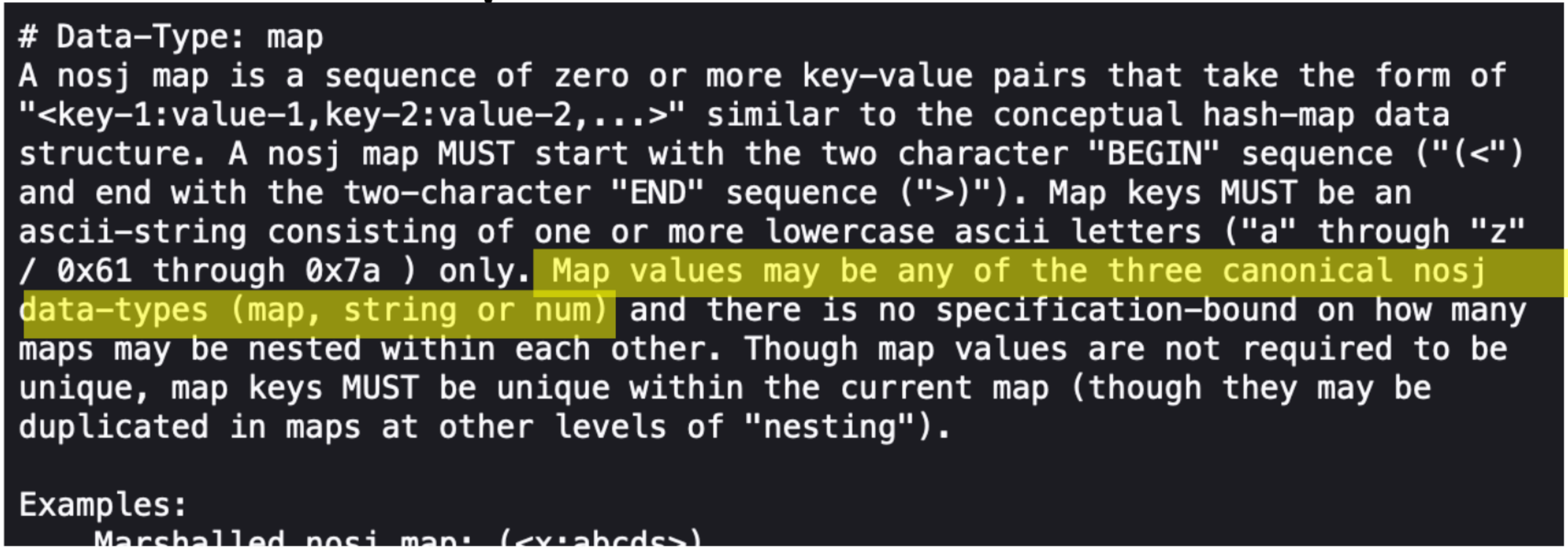
Input: `(<abc:defs>)`   Key: `“‘abc’’`



```
# Data-Type: map
A nosj map is a sequence of zero or more key-value pairs that take the form of
"<key-1:value-1,key-2:value-2,...>" similar to the conceptual hash-map data
structure. A nosj map MUST start with the two character "BEGIN" sequence ("(<")
and end with the two-character "END" sequence (">)"). Map keys MUST be an
ascii-string consisting of one or more lowercase ascii letters ("a" through "z"
/ 0x61 through 0x7a ) only. Map values may be any of the three canonical nosj
data-types (map, string or num) and there is no specification-bound on how many
maps may be nested within each other. Though map values are not required to be
unique, map keys MUST be unique within the current map (though they may be
duplicated in maps at other levels of "nesting").

Examples:
    Marshalled nosj map: (<x:abcds>)
```

Project 1A

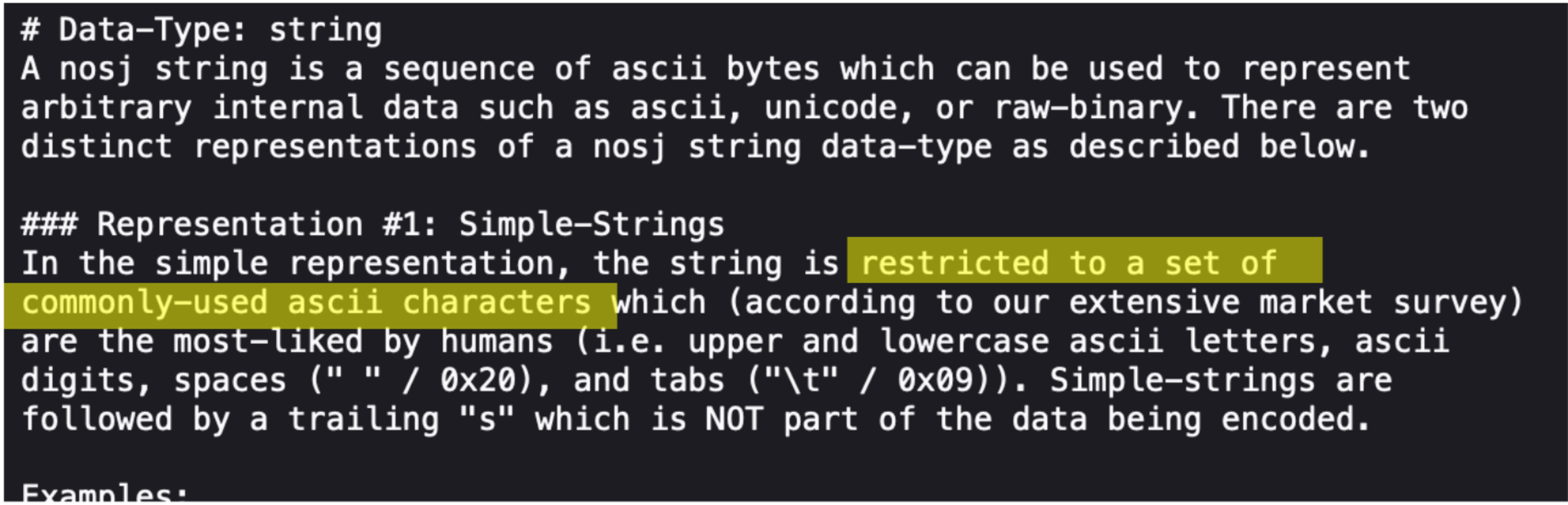
Input: `(<abc:defs>)`   Key: ‘‘abc’’



```
# Data-Type: map
A nosj map is a sequence of zero or more key-value pairs that take the form of
"<key-1:value-1,key-2:value-2,...>" similar to the conceptual hash-map data
structure. A nosj map MUST start with the two character "BEGIN" sequence ("(<")
and end with the two-character "END" sequence (">)"). Map keys MUST be an
ascii-string consisting of one or more lowercase ascii letters ("a" through "z"
/ 0x61 through 0x7a ) only. Map values may be any of the three canonical nosj
data-types (map, string or num) and there is no specification-bound on how many
maps may be nested within each other. Though map values are not required to be
unique, map keys MUST be unique within the current map (though they may be
duplicated in maps at other levels of "nesting").

Examples:
    Marshalled nosj map: (<x:abcds>)
```

Project 1A

Input: (<abc:defs>)   Key: “‘abc’’



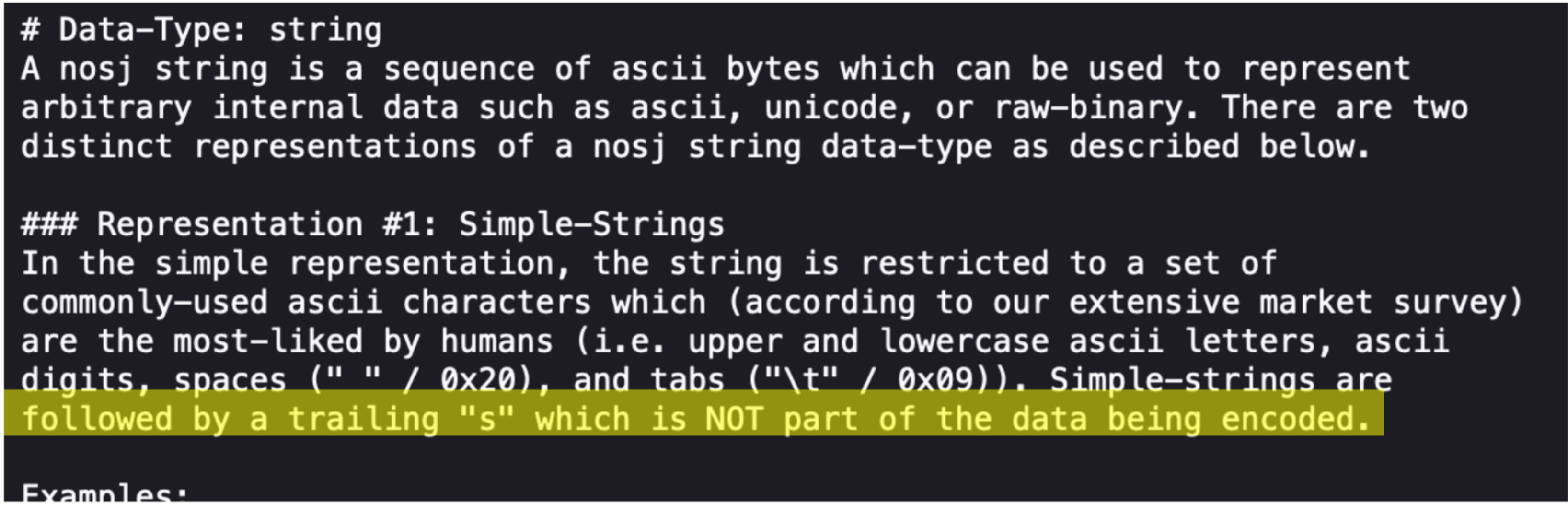
```
# Data-Type: string
A nosj string is a sequence of ascii bytes which can be used to represent
arbitrary internal data such as ascii, unicode, or raw-binary. There are two
distinct representations of a nosj string data-type as described below.

### Representation #1: Simple-Strings
In the simple representation, the string is restricted to a set of
commonly-used ascii characters which (according to our extensive market survey)
are the most-liked by humans (i.e. upper and lowercase ascii letters, ascii
digits, spaces (" " / 0x20), and tabs ("\t" / 0x09)). Simple-strings are
followed by a trailing "s" which is NOT part of the data being encoded.

Examples:
```

Project 1A

Input: (<abc:defs>)

Key: ‘‘abc’’

Value: ‘‘def’’



```
# Data-Type: string
A nosj string is a sequence of ascii bytes which can be used to represent
arbitrary internal data such as ascii, unicode, or raw-binary. There are two
distinct representations of a nosj string data-type as described below.

### Representation #1: Simple-Strings
In the simple representation, the string is restricted to a set of
commonly-used ascii characters which (according to our extensive market survey)
are the most-liked by humans (i.e. upper and lowercase ascii letters, ascii
digits, spaces (" " / 0x20), and tabs ("\t" / 0x09)). Simple-strings are
followed by a trailing "s" which is NOT part of the data being encoded.

Examples:
```

Project 1A Pro-Tips

- Don’t focus on what your code *should* be doing, focus on what your code *can be fed*
- Apply Software Engineering principles
  - Unit-testing, isolated responsibilities, etc.
- You ***can not*** patch/re-use a JSON parser
- You **can** use built-in libraries in your code

- **READ THE SPEC AGAIN**

Computer and Network Security

Lecture 03:
Hashing and Integrity

COMP-5370/6370
Fall 2024