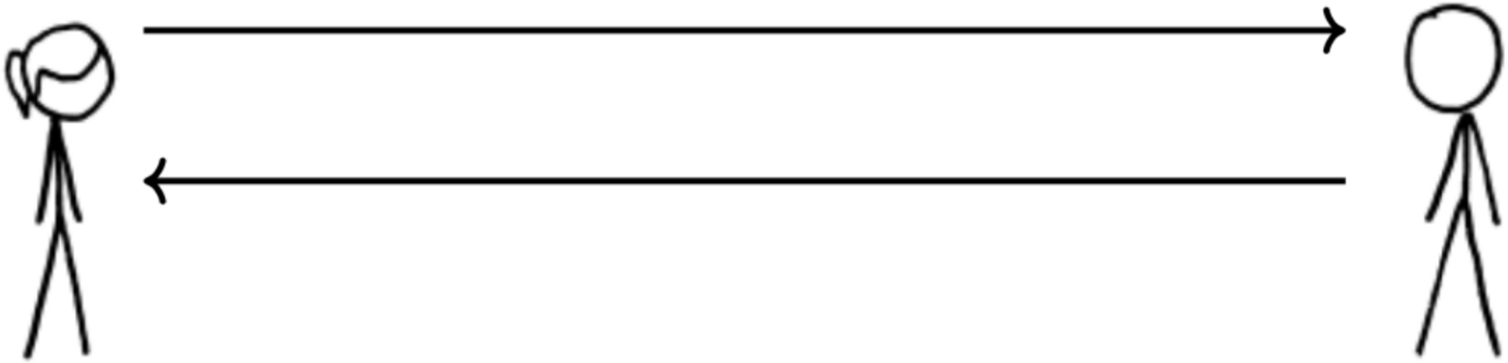
Computer and Network Security

Lecture 02:
Intro to Cryptography

COMP-5370/6370
Fall 2025

A Communications Channel

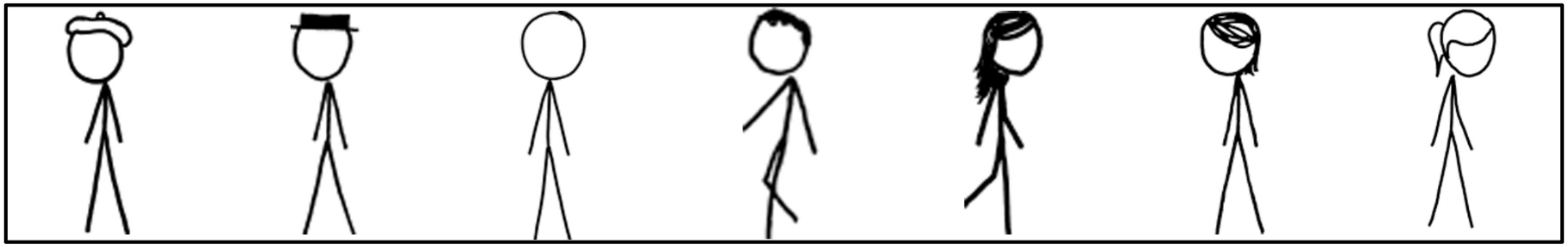


Classification of Actors

- **First Party (1P)**
  - Is knowingly and intentionally present in a conversation
- **Second Party (2P)**
  - Is knowingly and intentionally involved but not participating or present
- **Third Party (3P)**
  - Is unknowingly and *unintentionally* present in a conversation
- **Fourth Party (4P)** is “3P of a 3P”

Security Archetypes

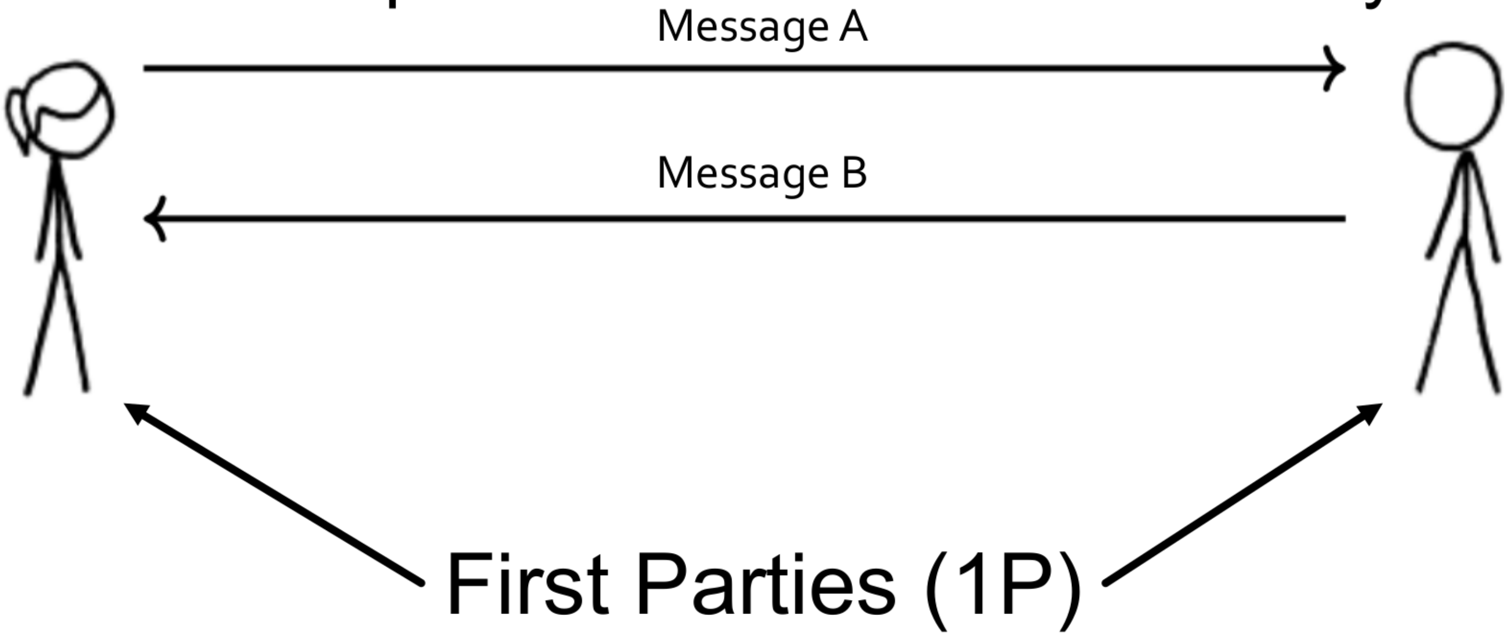
A **security archetype** is a named actor that is used as a representative of a nuanced actors and their capabilities/intentions.



- CS-101 equivalent: “foo”, “bar”, “baz”
- Are not 100% set-in-stone
  - Context, speaker, audience are all factors

Alice and Bob

- Normal, benign users trying to communicate with each other
- Pass **messages** back and forth
  - Context-specific details abstracted away

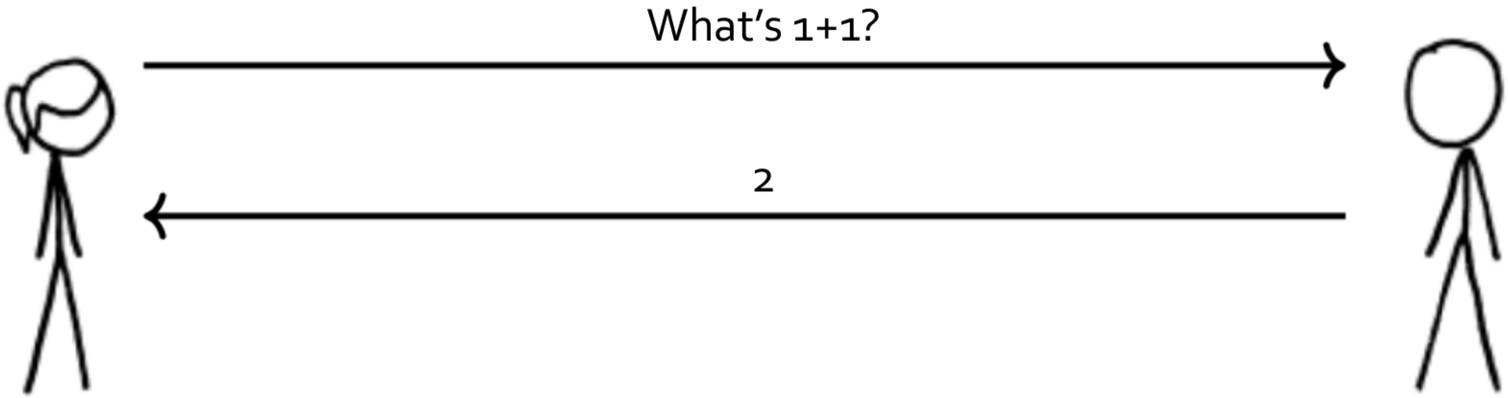


Message A

Message B

First Parties (1P)

A Communications Channel



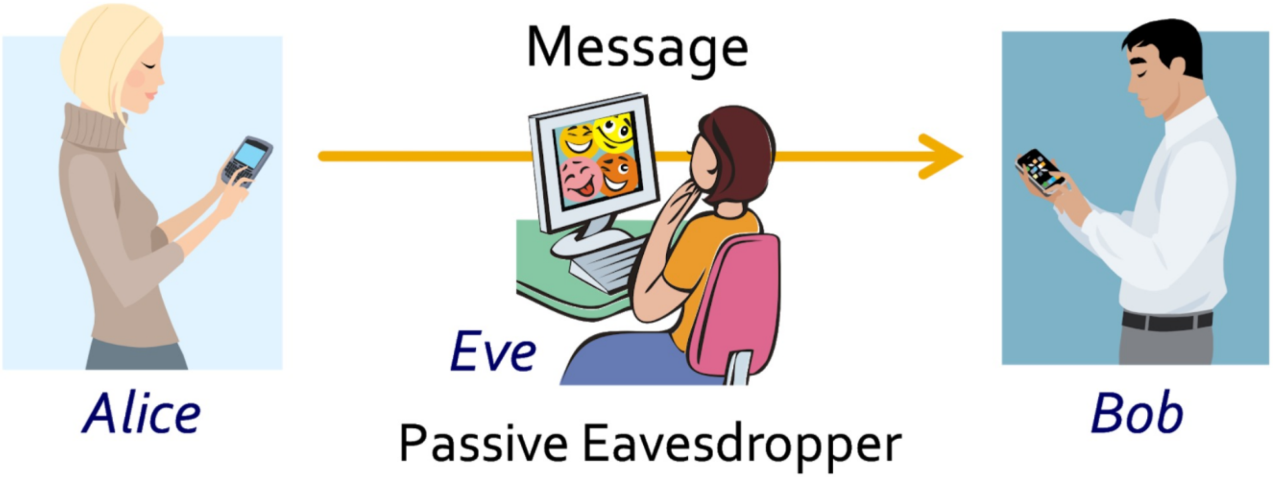
What’s 1+1?

2

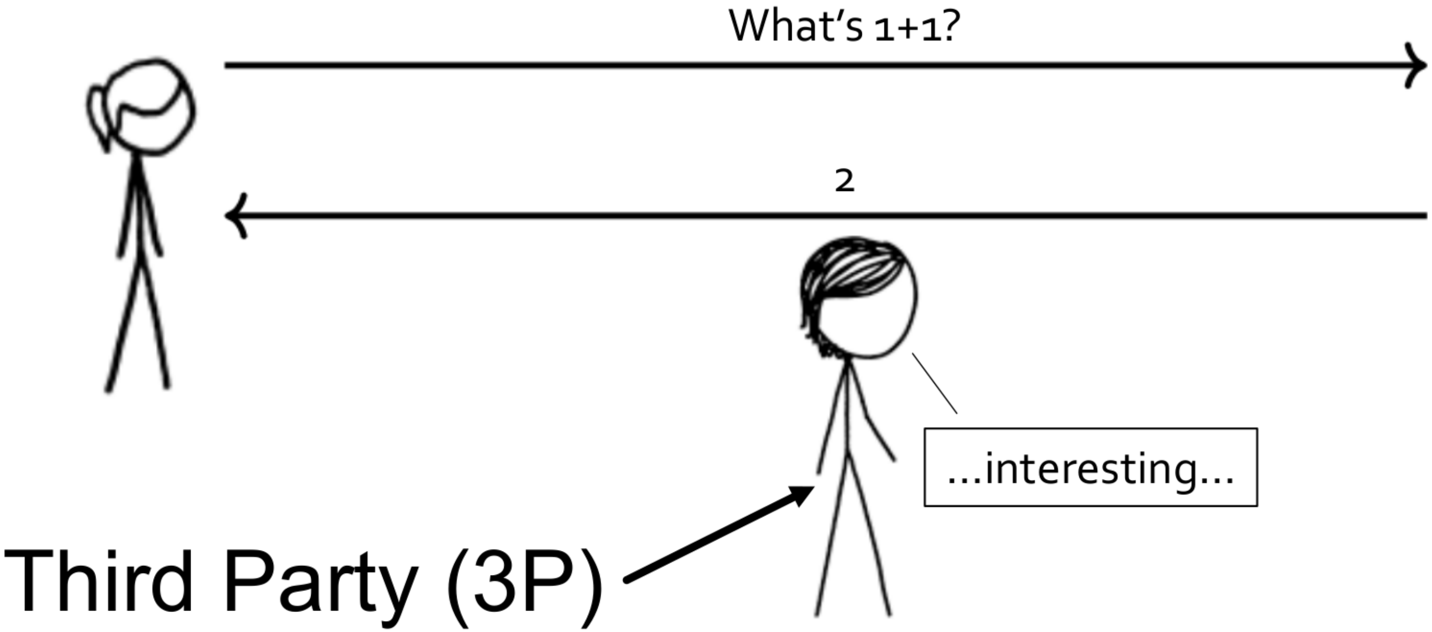
A Communications Channel

Eve the Eavesdropper

- A **passive** but malicious actor
- Can read messages but **cannot** modify, delay, discard, etc.



A Communications Channel



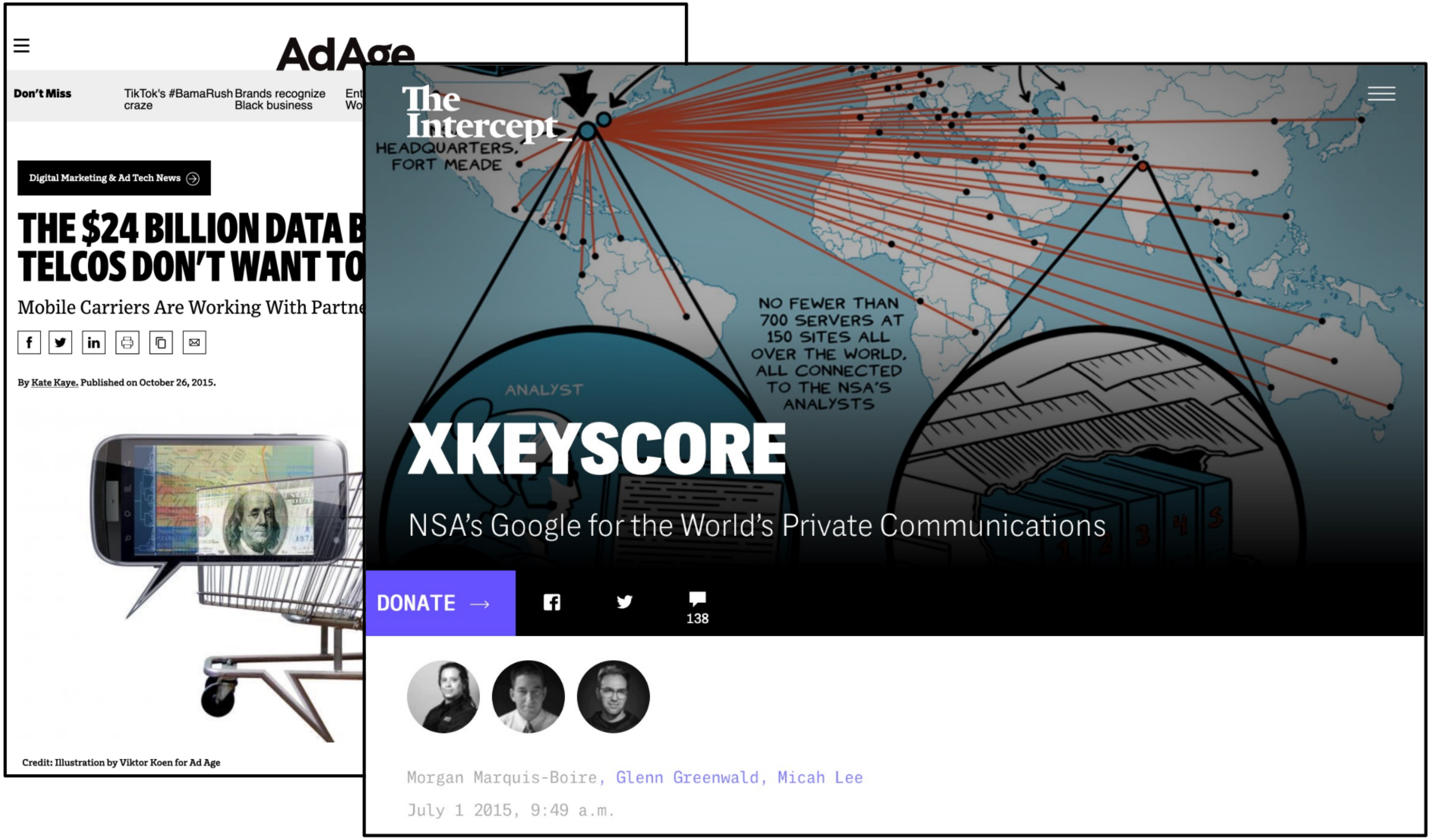
What’s 1+1?

2

...interesting...

Third Party (3P)

Foreshadowing



AdAge

Don't Miss

TikTok's #BamaRush craze

Brands recognize Black business

Ent Wo

Digital Marketing & Ad Tech News

THE $24 BILLION DATA B TELCOS DON'T WANT TO

Mobile Carriers Are Working With Partne

By Kate Kaye. Published on October 26, 2015.

Credit: Illustration by Viktor Koen for Ad Age

The Intercept_

HEADQUARTERS, FORT MEADE
NO FEWER THAN 700 SERVERS AT 150 SITES ALL OVER THE WORLD, ALL CONNECTED TO THE NSA'S ANALYSTS
ANALYST

XKEYSCORE

NSA's Google for the World's Private Communications

Morgan Marquis-Boire, Glenn Greenwald, Micah Lee

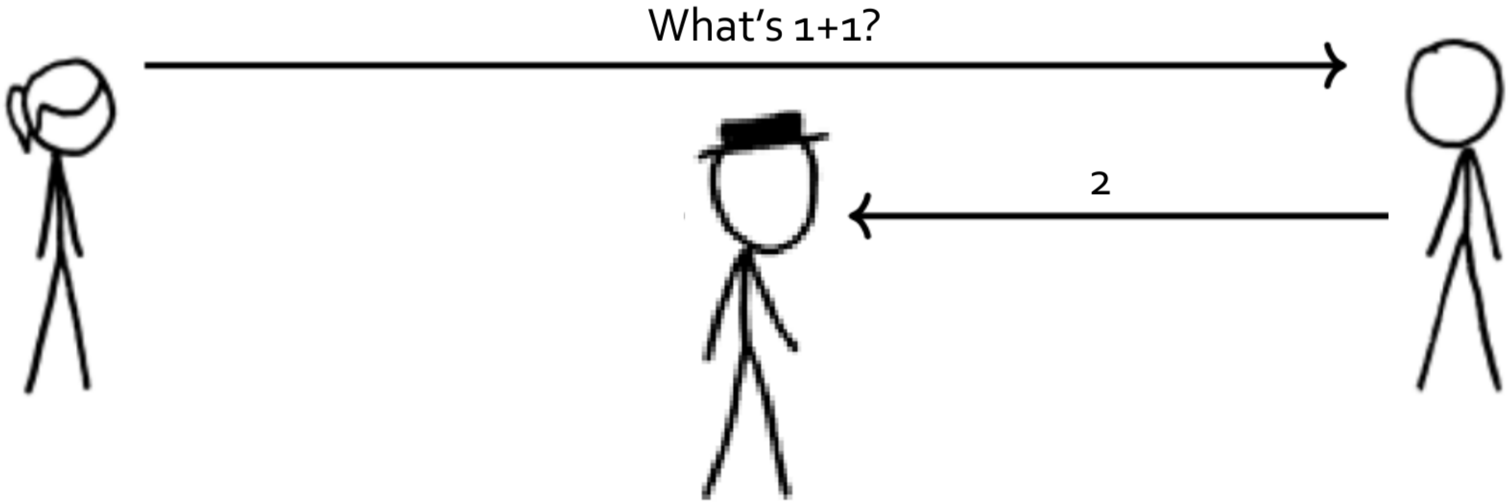
July 1 2015, 9:49 a.m.

Malicious Mallory

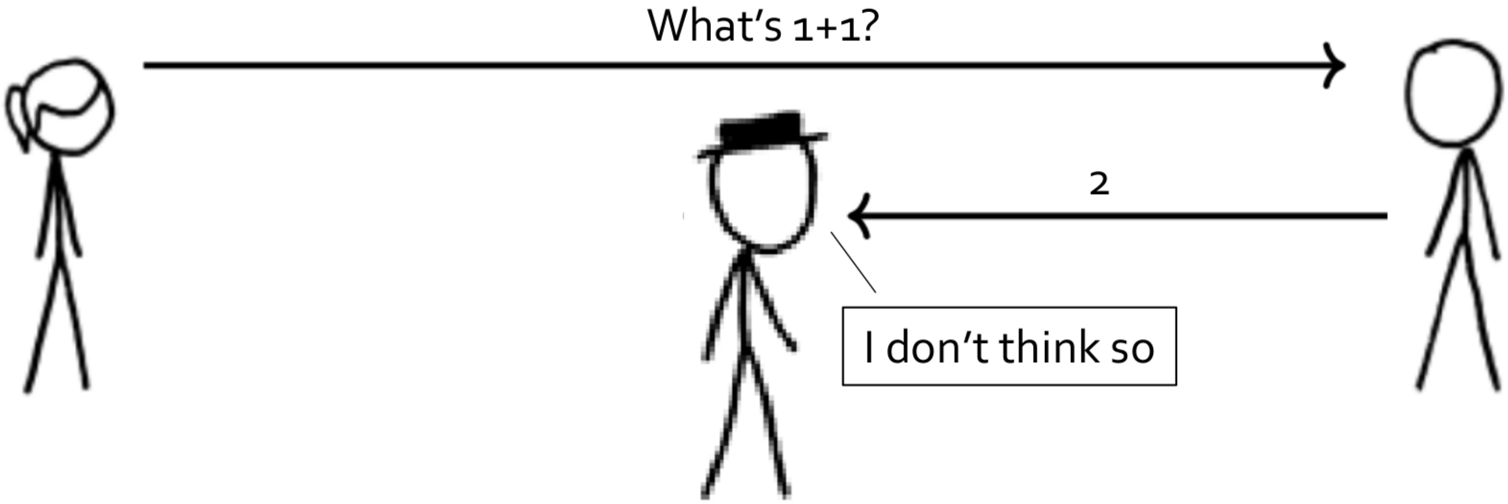
- An **active** and malicious actor
  - Has all passive capabilities (read messages)
  - Can modify/delay/discard messages
  - Can be an unintended end-point (MitM attack)

Mallory
Man-in-the-Middle

A Communications Channel



A Communications Channel

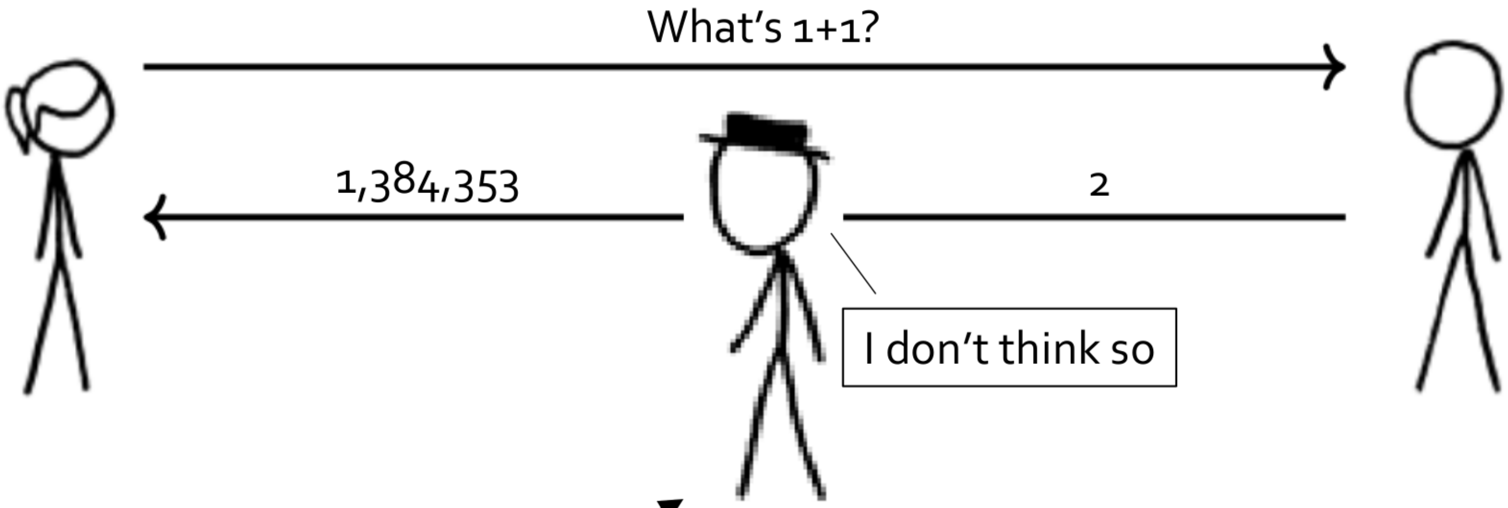


What’s 1+1?

2

I don’t think so

A Communications Channel



What’s 1+1?

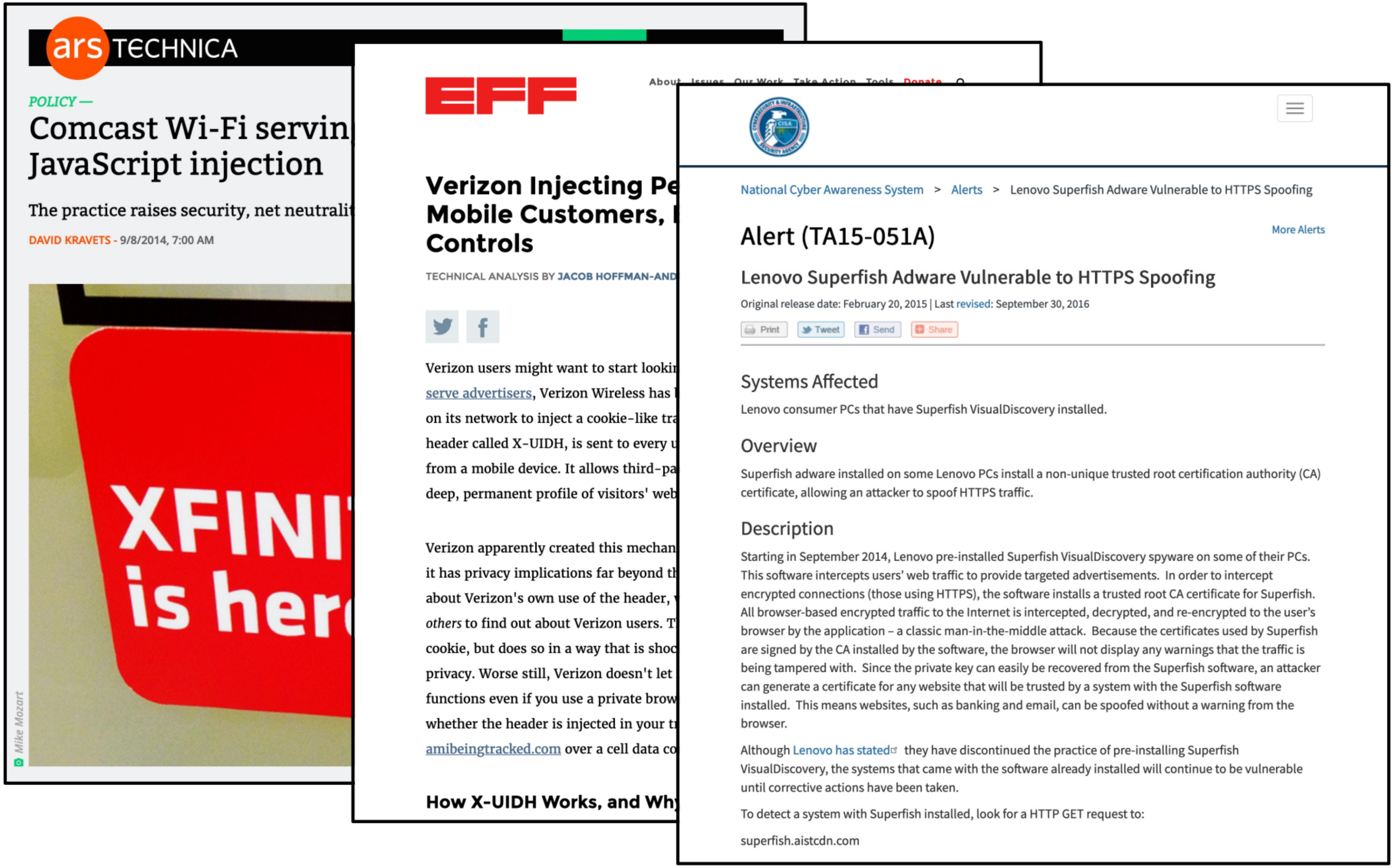
1,384,353

2

I don’t think so

Third Party (3P)

More Foreshadowing



ars TECHNICA

POLICY —

Comcast Wi-Fi servin
JavaScript injection

The practice raises security, net neutrali

DAVID KRAVETS - 9/8/2014, 7:00 AM

EFF

Verizon Injecting Pe
Mobile Customers, I
Controls

TECHNICAL ANALYSIS BY JACOB HOFFMAN-AND

Verizon users might want to start lookir
serve advertisers, Verizon Wireless has
on its network to inject a cookie-like tra
header called X-UIDH, is sent to every u
from a mobile device. It allows third-pa
deep, permanent profile of visitors' web

Verizon apparently created this mechan
it has privacy implications far beyond th
about Verizon's own use of the header, v
others to find out about Verizon users. T
cookie, but does so in a way that is shoc
privacy. Worse still, Verizon doesn't let
functions even if you use a private brow
whether the header is injected in your t
amibeingtracked.com over a cell data co

How X-UIDH Works, and Why

National Cyber Awareness System > Alerts > Lenovo Superfish Adware Vulnerable to HTTPS Spoofing

Alert (TA15-051A)

More Alerts

Lenovo Superfish Adware Vulnerable to HTTPS Spoofing

Original release date: February 20, 2015 | Last revised: September 30, 2016

Systems Affected

Lenovo consumer PCs that have Superfish VisualDiscovery installed.

Overview

Superfish adware installed on some Lenovo PCs install a non-unique trusted root certification authority (CA) certificate, allowing an attacker to spoof HTTPS traffic.

Description

Starting in September 2014, Lenovo pre-installed Superfish VisualDiscovery spyware on some of their PCs. This software intercepts users' web traffic to provide targeted advertisements. In order to intercept encrypted connections (those using HTTPS), the software installs a trusted root CA certificate for Superfish. All browser-based encrypted traffic to the Internet is intercepted, decrypted, and re-encrypted to the user's browser by the application – a classic man-in-the-middle attack. Because the certificates used by Superfish are signed by the CA installed by the software, the browser will not display any warnings that the traffic is being tampered with. Since the private key can easily be recovered from the Superfish software, an attacker can generate a certificate for any website that will be trusted by a system with the Superfish software installed. This means websites, such as banking and email, can be spoofed without a warning from the browser.

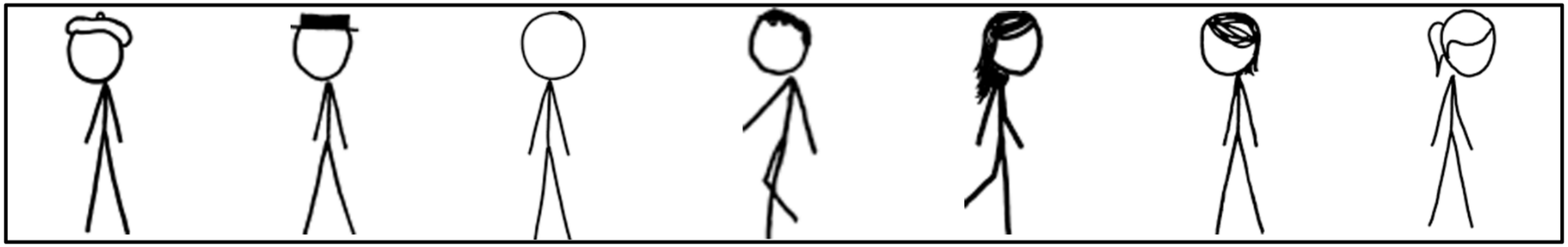
Although Lenovo has stated they have discontinued the practice of pre-installing Superfish VisualDiscovery, the systems that came with the software already installed will continue to be vulnerable until corrective actions have been taken.

To detect a system with Superfish installed, look for a HTTP GET request to:

superfish.aistcdn.com

Security Archetypes

A **security archetype** is a named actor that is used as a representative of a nuanced actors and their capabilities/intentions.



- CS-101 equivalent: “foo”, “bar”, “baz”
- Are not 100% set-in-stone
  - Context, speaker, audience are all factors
- More can be defined based on needs

Classification of Actors

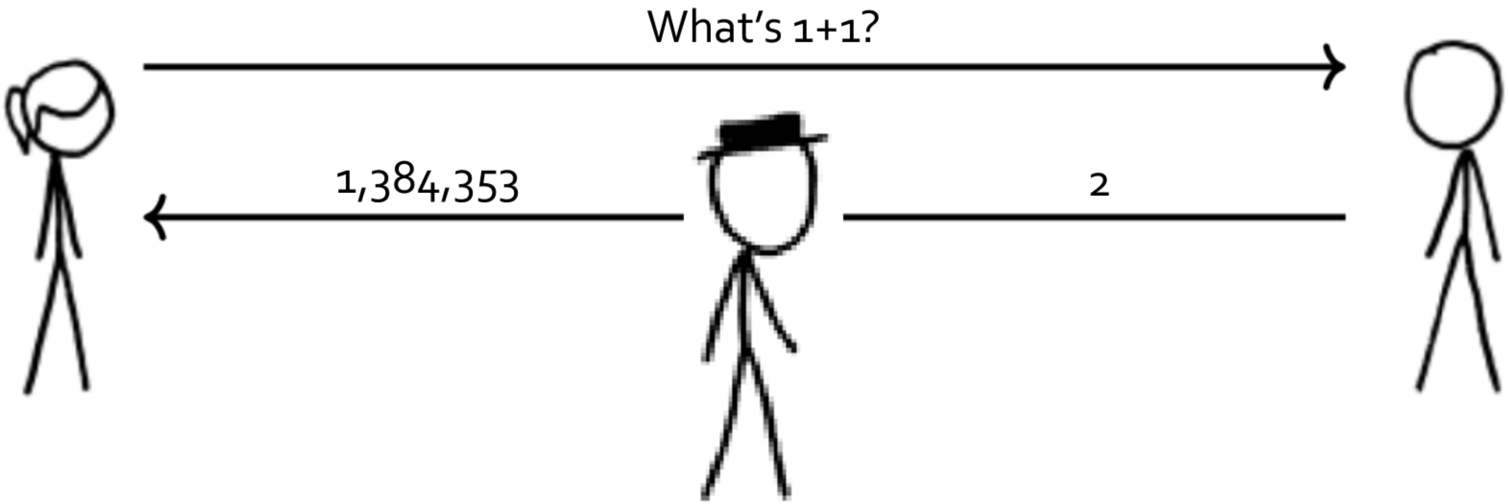
- **First Party (1P)**
  - Is knowingly and intentionally present in a conversation (Alice/Bob)
- **Second Party (2P)**
  - Is knowingly and intentionally involved but not participating or present
- **Third Party (3P)**
  - Is unknowingly and *unintentionally* present in a conversation (Eve/Mallory)
- **Fourth Party (4P)** is “3P of a 3P”

Classification of Actors

- **First Party (1P)**
  - Is knowingly and intentionally present in a conversation (Alice/Bob)
- **Second Party (2P)**
  - Is knowingly and intentionally involved but not participating or present
- **Third Party (3P)**
  - Is unknowingly and *unintentionally* present in a conversation (Eve/Mallory)
- **Fourth Party (4P)** is “3P of a 3P”

Classification of Abilities

- **Passive Actor**
  - Has the ability to *look* but not *touch*
- **Active Actor**
  - Has the ability to look *and* touch



What’s 1+1?

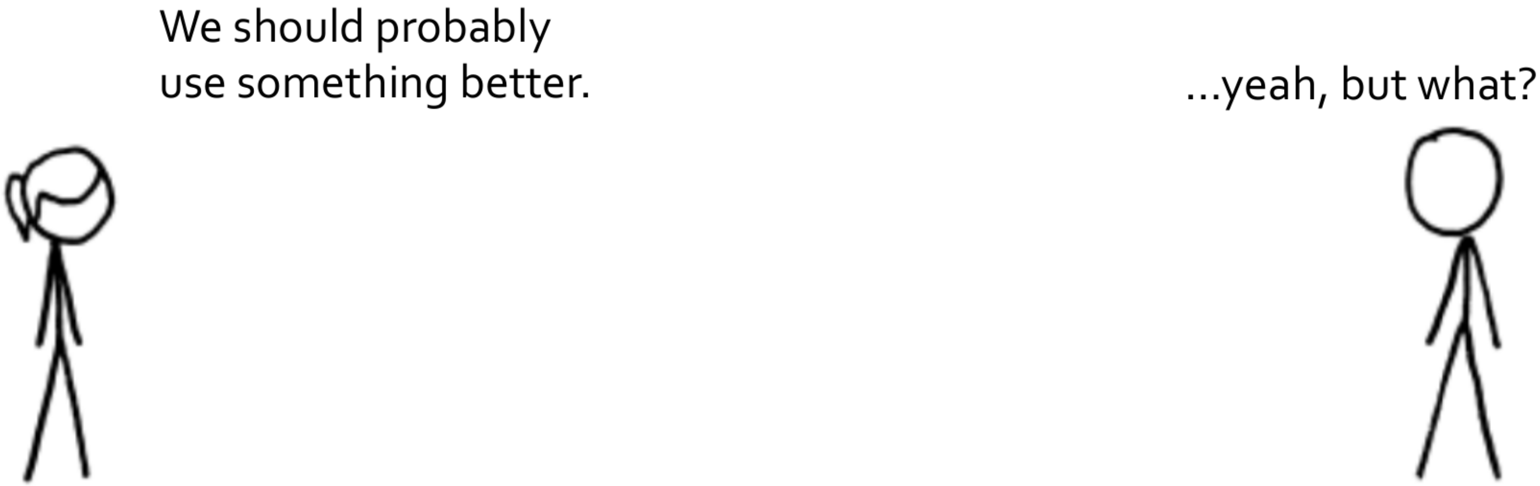
1,384,353

2

Classification of Abilities

- **Passive Actor**
  - Has the ability to *look* but not *touch*
- **Active Actor**
  - Has the ability to look *and* touch
- **Man-in-the-Middle Actor (MitM)**
  - Sub-class of Active Attacker
  - Usually requires “on path” vantage point

A Communications Channel



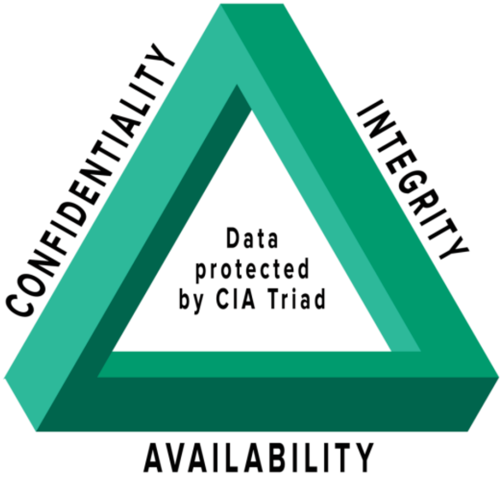
What is desired?

What do you want when talking to your friends and family?

What do you want when talking to your doctor?

What do you want when talking to Comcast, University, other org/company?

Classical CIA Triad



- Canonical security properties (baseline)
- You will see this **EVERYWHERE**
- It not bad but it’s overly vague and can be interpreted many different ways

Security Properties

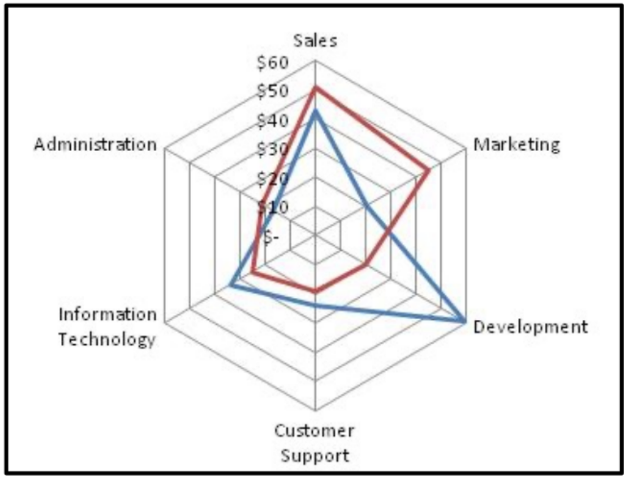
- Confidentiality
- Availability
- Integrity
- Resiliency
- Authentication
- Nonrepudiation
- Forward-Secret
- Authenticity
- Anonymity
- *Many, many more*

- Characterize system at abstract level
- Make it easy to describe protections provided
- Make it easy to describe protections **not** provided

Security Properties

- Confidentiality
- Availability
- Integrity
- Resiliency
- Authentication
- Nonrepudiation
- Forward-Secret
- Authenticity
- Anonymity
- *Many, many more*

- Some properties make others harder
- Sometimes, have to make trade-offs



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

Cryptography

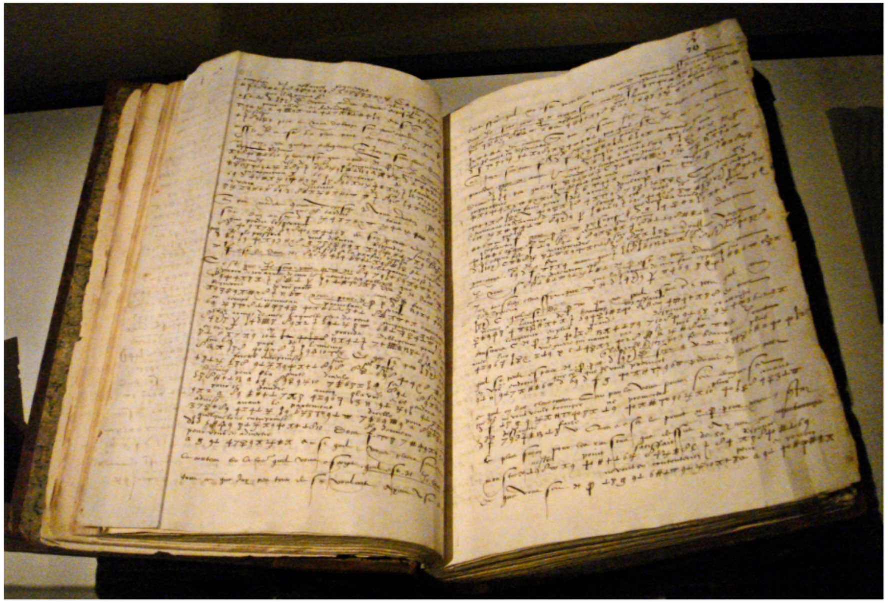
- Used for millennia to protect comms.

Cipher Disk

Cryptography

- Used for millennia to protect comms.

- Comes in many forms/constructions



Cryptography

- Used for millennia to protect comms.
- Comes in many forms/constructions
- Until the Internet, was largely unused by normal people

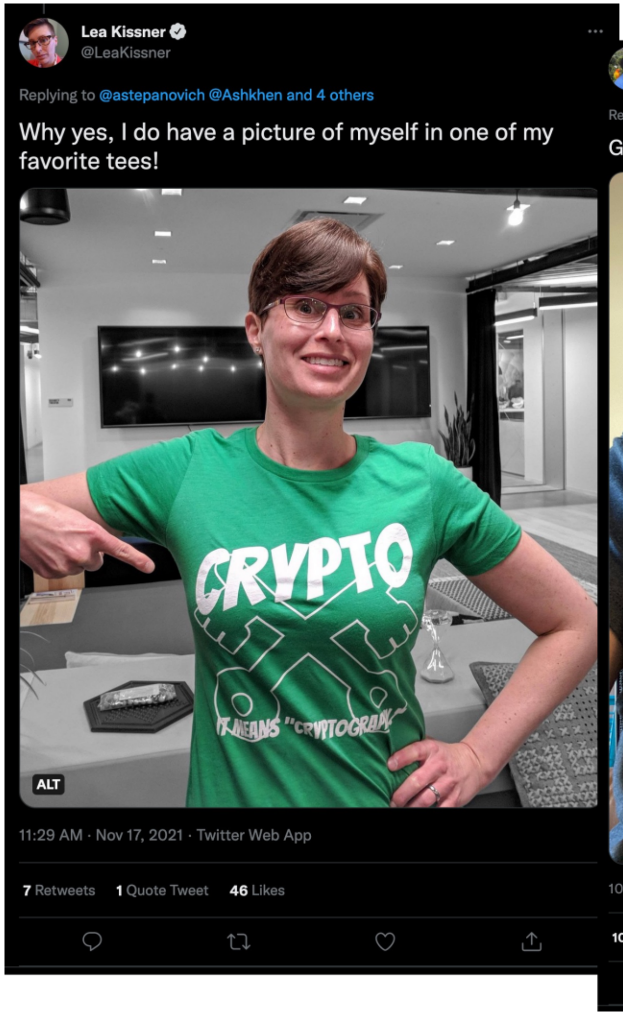
Enigma Machine

Don’t Forget…

“Crypto” means Cryptography!

So sayth Shady Professor

...and everyone else



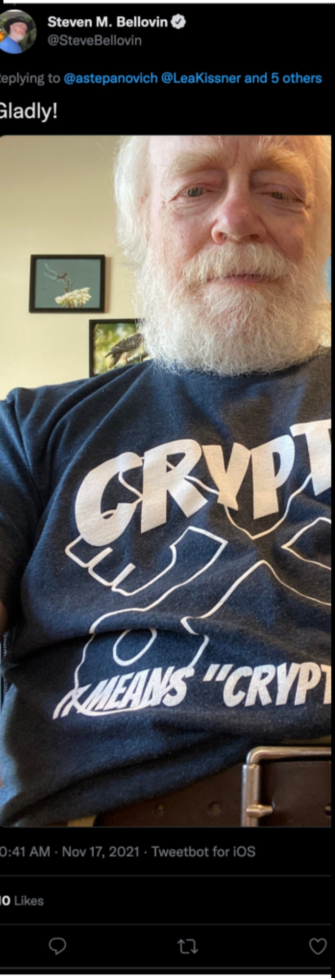
Lea Kissner
@LeaKissner

Replying to @astepanovich @Ashkhen and 4 others

Why yes, I do have a picture of myself in one of my favorite tees!

11:29 AM · Nov 17, 2021 · Twitter Web App

7 Retweets 1 Quote Tweet 46 Likes



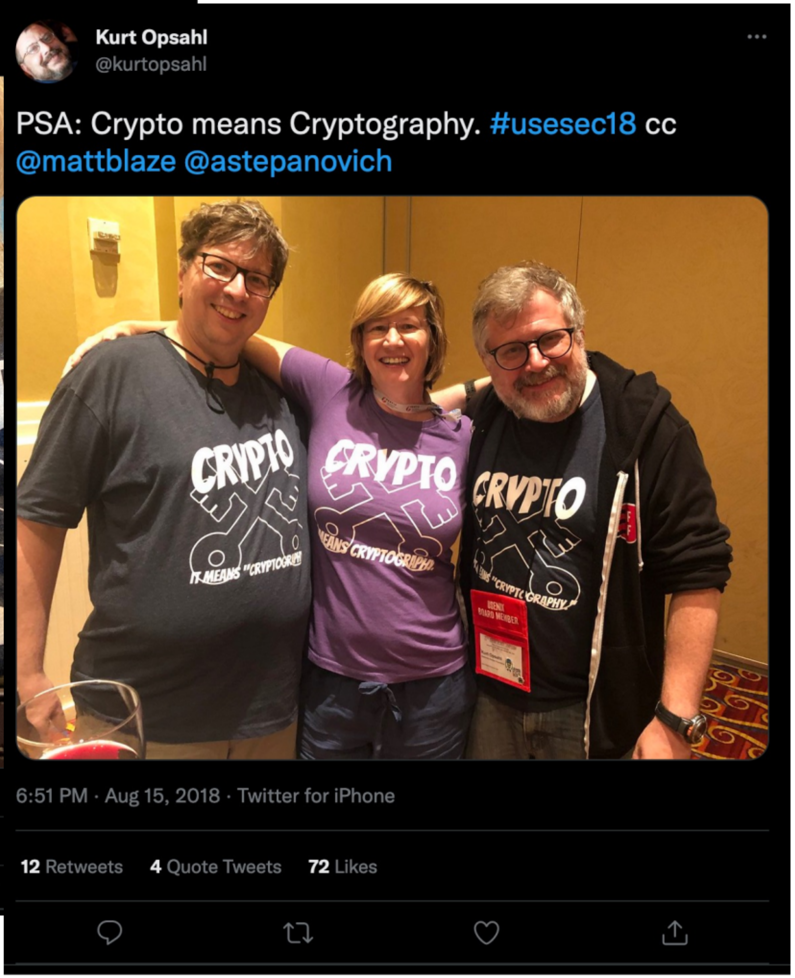
Steven M. Bellovin
@SteveBellovin

Replying to @astepanovich @LeaKissner and 5 others

Gladly!

10:41 AM · Nov 17, 2021 · Tweetbot for iOS

10 Likes



Kurt Opsahl
@kurtopsahl

PSA: Crypto means Cryptography. #usesec18 cc @mattblaze @astepanovich

6:51 PM · Aug 15, 2018 · Twitter for iPhone

12 Retweets 4 Quote Tweets 72 Likes

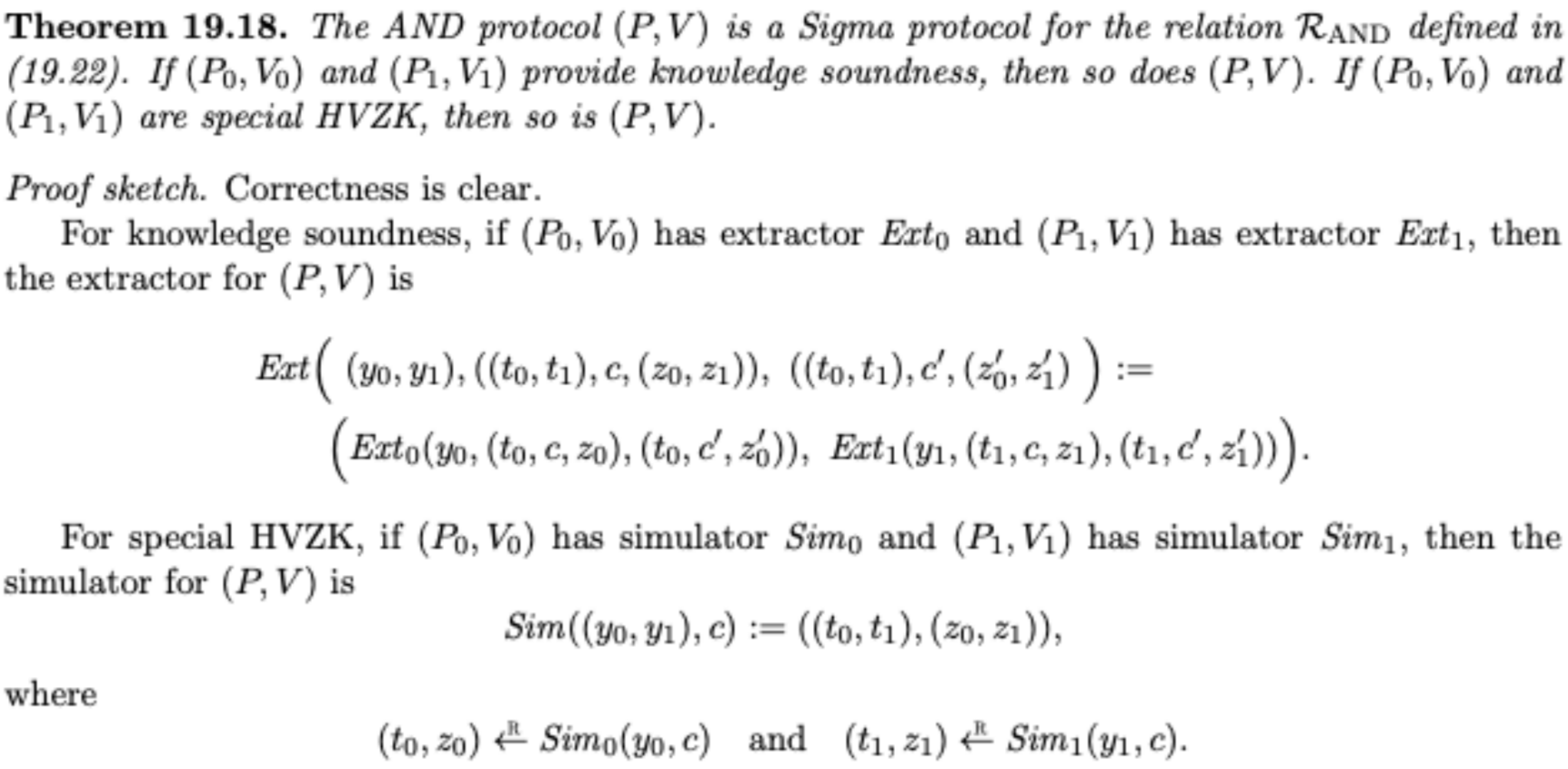
WARNING

I AM NOT A CRYPTOGRAPHER

WARNING

YOU ARE NOT A CRYPTOGRAPHER

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

$$\begin{aligned} &\mathit{Ext}\Big(\ (y_0, y_1), ((t_0, t_1), c, (z_0, z_1)),\ \ ((t_0, t_1), c', (z_0', z_1')\ \Big) := \\ &\qquad \Big(\mathit{Ext}_0(y_0, (t_0, c, z_0), (t_0, c', z_0')),\ \ \mathit{Ext}_1(y_1, (t_1, c, z_1), (t_1, c', z_1'))\Big). \end{aligned}$$

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

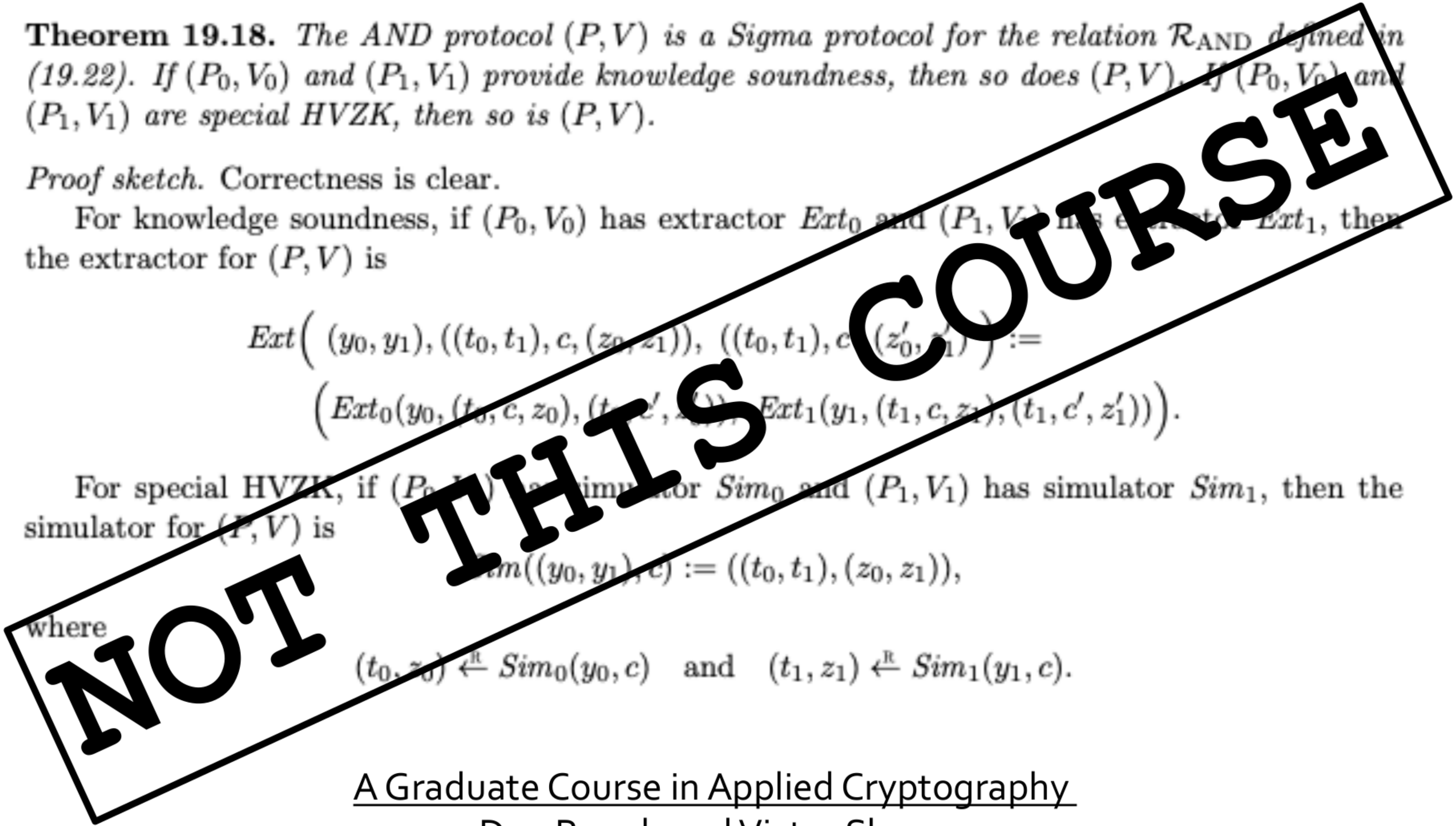
$$\mathit{Sim}((y_0, y_1), c) := ((t_0, t_1), (z_0, z_1)),$$

where

$$(t_0, z_0) \xleftarrow{\text{R}} \mathit{Sim}_0(y_0, c) \quad \text{and} \quad (t_1, z_1) \xleftarrow{\text{R}} \mathit{Sim}_1(y_1, c).$$

A Graduate Course in Applied Cryptography
Dan Boneh and Victor Shoup
https://toc.cryptobook.us/

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

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

NOT THIS COURSE

A Graduate Course in Applied Cryptography
Dan Boneh and Victor Shoup
https://toc.cryptobook.us/

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

What is Cryptography?

Cryptography Is

- Fundamental tool of security & privacy
- Extremely well studied by math-ppl
- A tool in the toolbox to use when useful
- A whole lot of fun to use and break

Cryptography Is Not

- An S&P cure-all
- Reliable unless expertly implemented it
- Reliable unless deeply reviewed and tested
- Something you can learn in a weekend, month, or year

Randomness

Random data is unpredictable bits to the attacker without any pattern or structure.

- Any bit has exactly the same chance of:
  - Being 0 (50%)
  - Being 1 (50%)

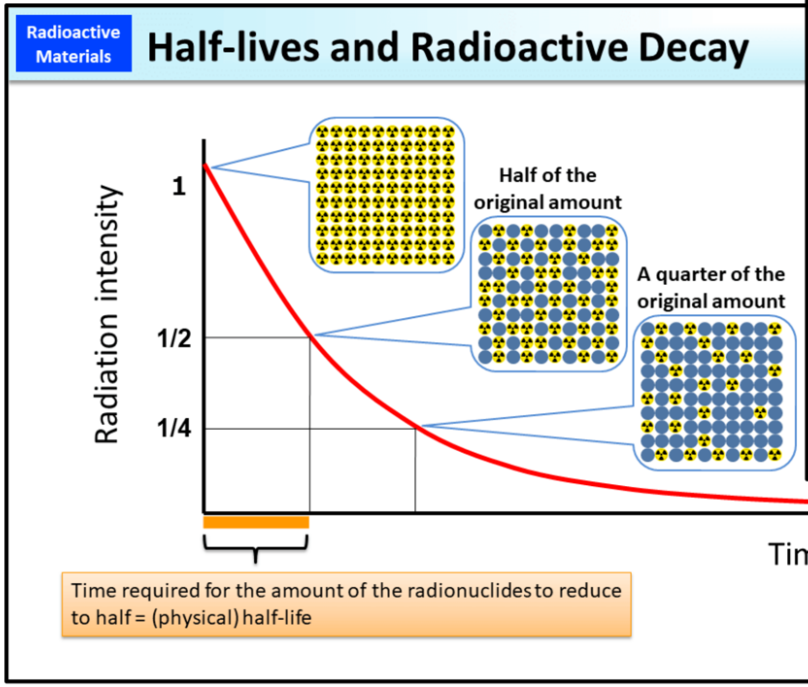
- **Computers are really bad at randomness**

True Randomness

True random data can not be created, it can only be measured from an external physical process.

True Randomness

True random data can not be created, it can only be measured from an external physical process.



Radioactive Materials

Half-lives and Radioactive Decay

Time required for the amount of the radionuclides to reduce to half = (physical) half-life

True Randomness

True random data can not be created, it can only be measured from an external physical process.

- **Must be measured in secret**
- Is extremely slow and scarce

- **Makes fast computer slow**

Pseudorandomness

Pseudorandom data mimics the properties of random data but is deterministically generated based on an input.

- Computers are very good at doing very tedious things very quickly
- Less “random” than true randomness
- More achievable than true randomness

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**

CSPRNG

A **Cryptographically Secure Pseudorandom Number Generator (CSPRNG)** maps a k-bit random input to arbitrary-length pseudorandom outputs.

- Only trustworthy way to generate arbitrary amounts of randomness from a seed
- **Safe for generating keys and all other randomness needed for cryptography**

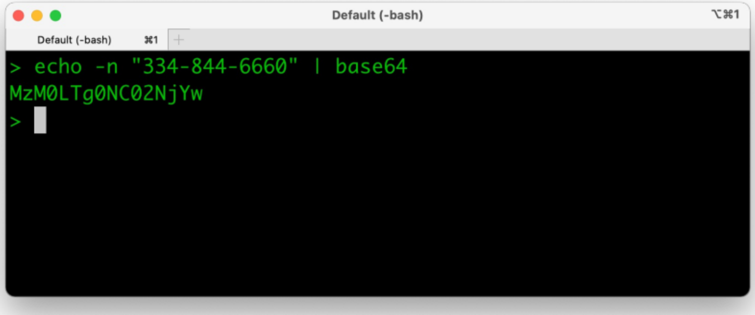
Canonical Randomness Sources

- `/dev/urandom`
  - Kinda-random source of bytes
  - Always generates data even if they're not-really-random (i.e. non-blocking device)

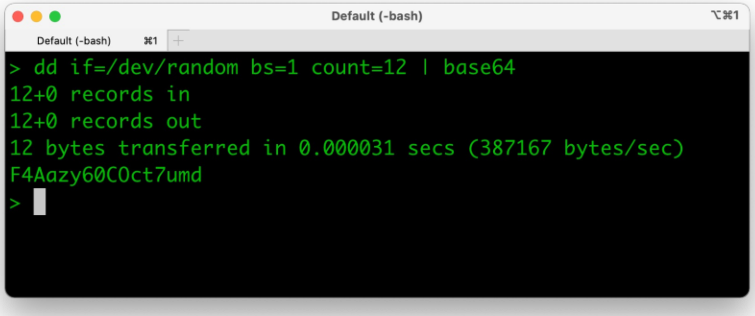
- `/dev/random`
  - Really random source of data
  - Pauses if can not safely generate more data at current time (i.e. blocking device)

Real-World Randomness

MzM0LTg0NC02NjYw F4azy60COct7umd



```
> echo -n "334-844-6660" | base64
MzM0LTg0NC02NjYw
> 
```

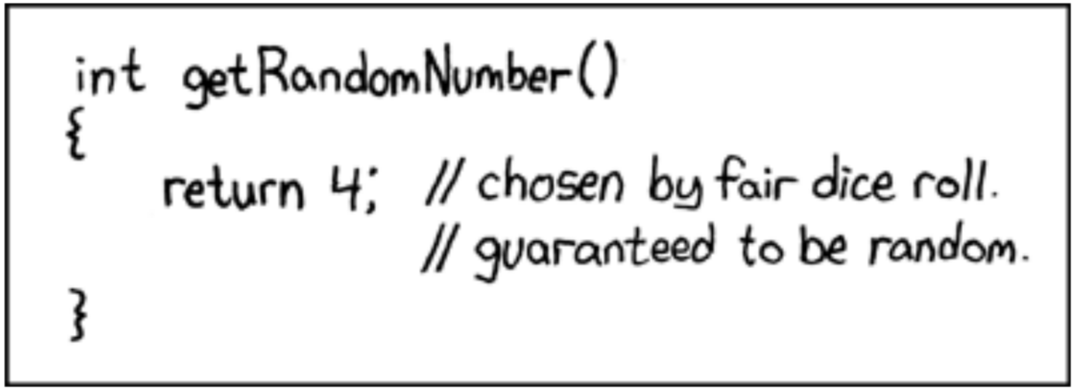


```
> dd if=/dev/random bs=1 count=12 | base64
12+0 records in
12+0 records out
12 bytes transferred in 0.000031 secs (387167 bytes/sec)
F4Aazy60COct7umd
> 
```

Real-World Randomness

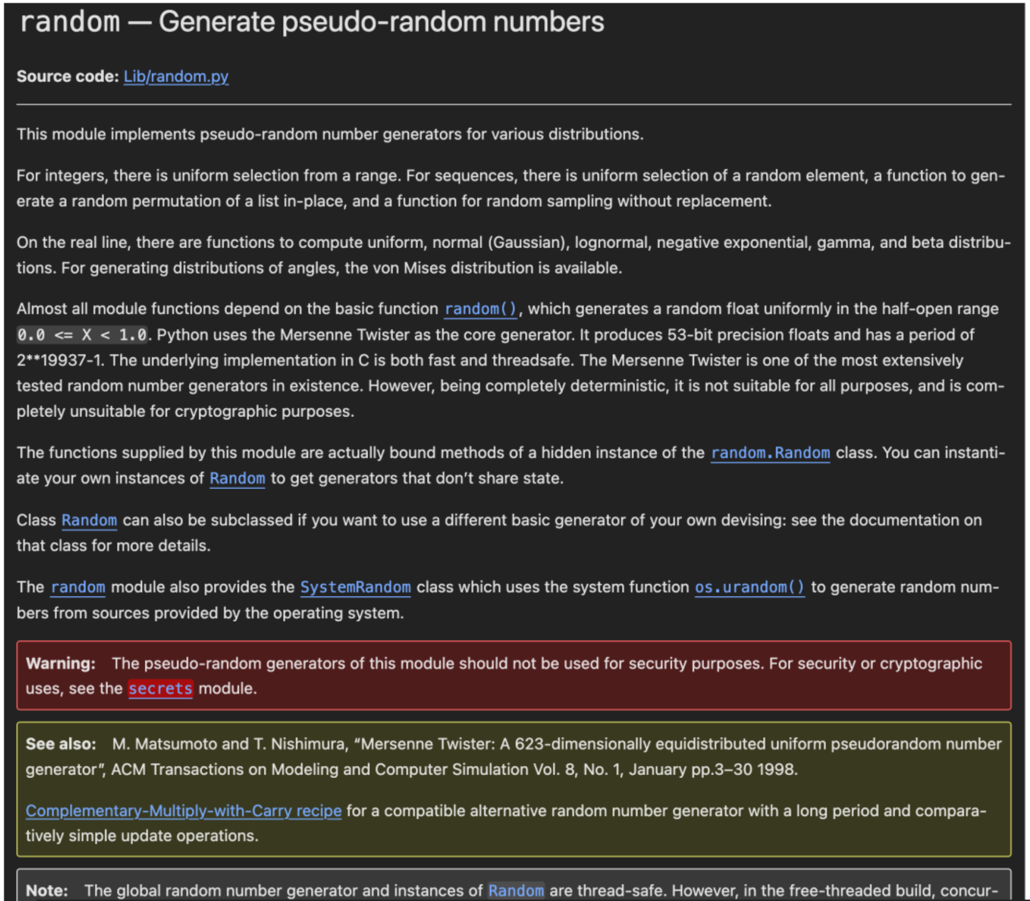
TOUR OF ACCOUNTING
OVER HERE WE HAVE OUR RANDOM NUMBER GENERATOR.
scottadams@aol.com
www.dilbert.com
NINE NINE NINE NINE NINE NINE
© 2001 United Feature Syndicate, Inc.
10/25/01
ARE YOU SURE THAT'S RANDOM?
THAT'S THE PROBLEM WITH RANDOMNESS: YOU CAN NEVER BE SURE.

“Random” isn’t always *random*



```
int getRandomNumber()
{
    return 4; // chosen by fair dice roll.
              // guaranteed to be random.
}
```

Not all "random" is *random*



random — Generate pseudo-random numbers

Source code: Lib/random.py

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

Almost all module functions depend on the basic function `random()`, which generates a random float uniformly in the half-open range `0.0 <= X < 1.0`. Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of 2**19937-1. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the `random.Random` class. You can instantiate your own instances of `Random` to get generators that don't share state.

Class `Random` can also be subclassed if you want to use a different basic generator of your own devising: see the documentation on that class for more details.

The `random` module also provides the `SystemRandom` class which uses the system function `os.urandom()` to generate random numbers from sources provided by the operating system.

Warning: The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the `secrets` module.

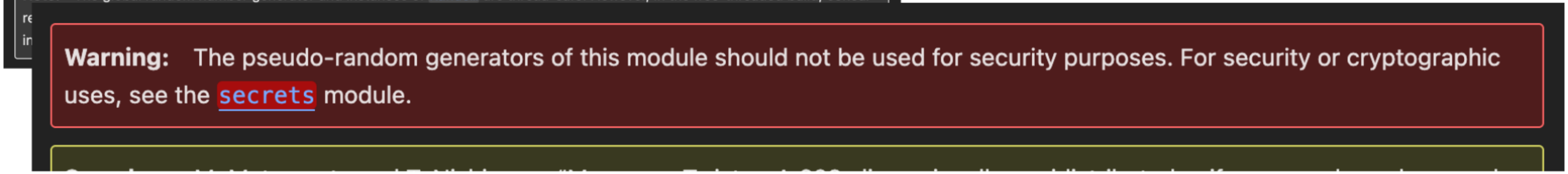
See also: M. Matsumoto and T. Nishimura, "Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator", ACM Transactions on Modeling and Computer Simulation Vol. 8, No. 1, January pp.3–30 1998.

Complementary-Multiply-with-Carry recipe for a compatible alternative random number generator with a long period and comparatively simple update operations.

Note: The global random number generator and instances of `Random` are thread-safe. However, in the free-threaded build, concur-

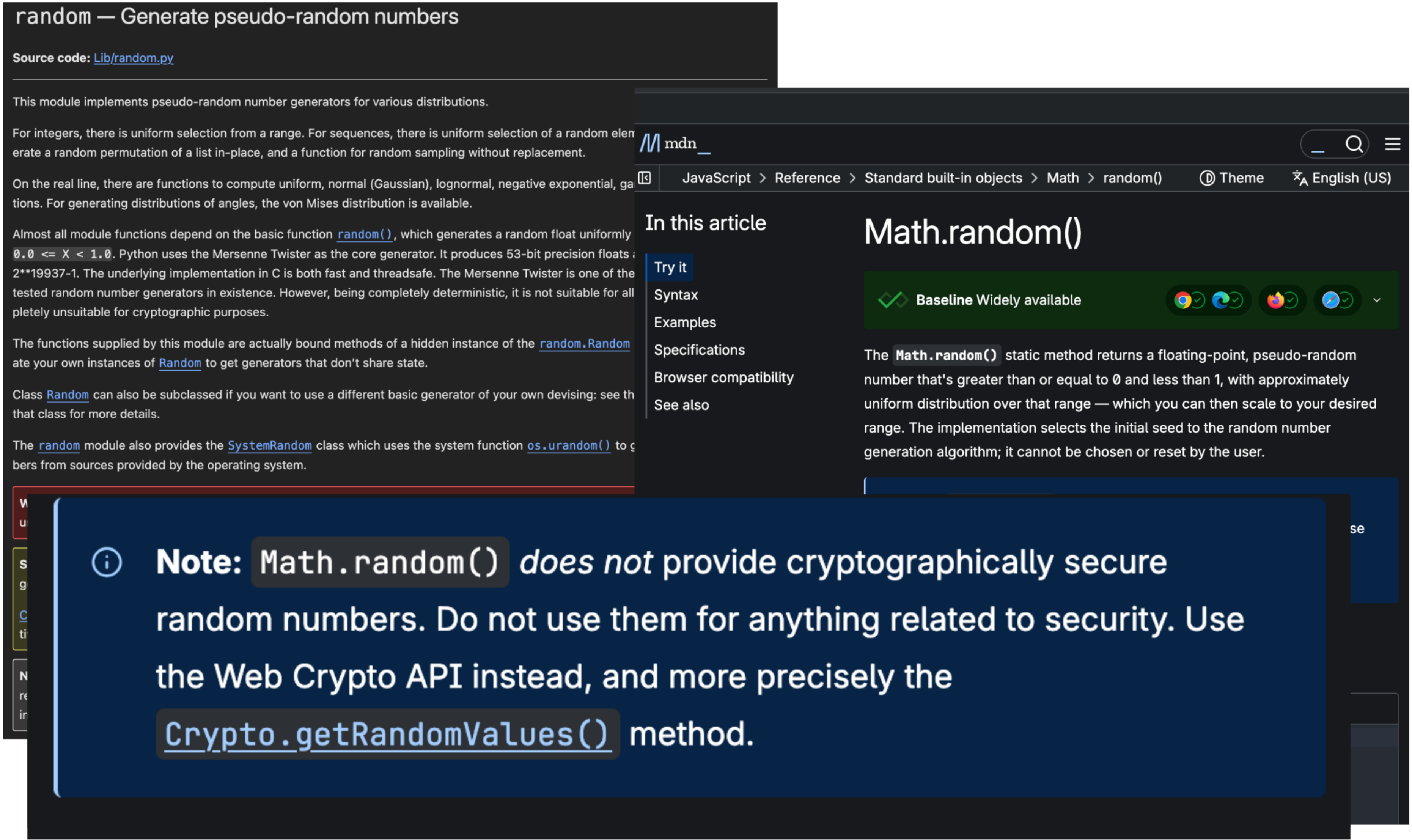
re

in



Warning: The pseudo-random generators of this module should not be used for security purposes. For security or cryptographic uses, see the `secrets` module.

Not all “random” is *random*



random — Generate pseudo-random numbers

Source code: Lib/random.py

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random elem erate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, ga tions. For generating distributions of angles, the von Mises distribution is available.

Almost all module functions depend on the basic function `random()`, which generates a random float uniformly `0.0 <= X < 1.0`. Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats a 2**19937-1. The underlying implementation in C is both fast and threadsafe. The Mersenne Twister is one of the tested random number generators in existence. However, being completely deterministic, it is not suitable for all pletely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the `random.Random` ate your own instances of `Random` to get generators that don’t share state.

Class `Random` can also be subclassed if you want to use a different basic generator of your own devising: see th that class for more details.

The `random` module also provides the `SystemRandom` class which uses the system function `os.urandom()` to g bers from sources provided by the operating system.

mdn_
JavaScript > Reference > Standard built-in objects > Math > random()
Theme
English (US)

In this article

- Try it
- Syntax
- Examples
- Specifications
- Browser compatibility
- See also

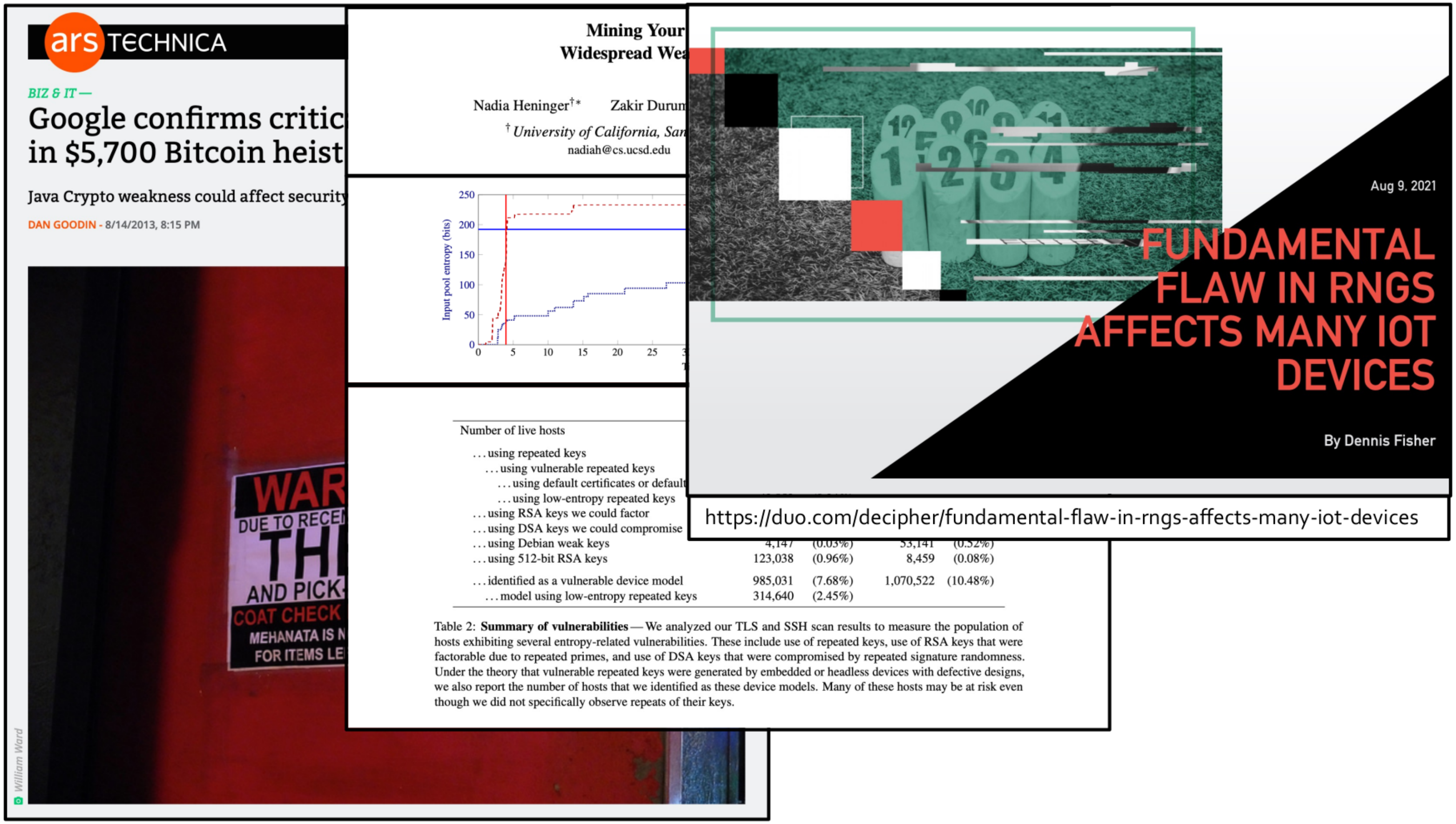
Math.random()

Baseline Widely available

The `Math.random()` static method returns a floating-point, pseudo-random number that's greater than or equal to `0` and less than 1, with approximately uniform distribution over that range — which you can then scale to your desired range. The implementation selects the initial seed to the random number generation algorithm; it cannot be chosen or reset by the user.

Note: `Math.random()` *does not* provide cryptographically secure random numbers. Do not use them for anything related to security. Use the Web Crypto API instead, and more precisely the `Crypto.getRandomValues()` method.

Poor Randomness Sources cause Major Vulnerabilities



BIZ & IT —

Google confirms critic
in $5,700 Bitcoin heist

Java Crypto weakness could affect security

DAN GOODIN - 8/14/2013, 8:15 PM

Mining Your
Widespread Wea

Nadia Heninger[†*] Zakir Durum
[†] *University of California, San*
nadiah@cs.ucsd.edu

Number of live hosts				
. . . using repeated keys				
. . . using vulnerable repeated keys				
. . . using default certificates or default				
. . . using low-entropy repeated keys				
. . . using RSA keys we could factor				
. . . using DSA keys we could compromise				
. . . using Debian weak keys	4,147	(0.03%)	53,141	(0.52%)
. . . using 512-bit RSA keys	123,038	(0.96%)	8,459	(0.08%)
. . . identified as a vulnerable device model	985,031	(7.68%)	1,070,522	(10.48%)
. . . model using low-entropy repeated keys	314,640	(2.45%)		

Table 2: **Summary of vulnerabilities** — We analyzed our TLS and SSH scan results to measure the population of hosts exhibiting several entropy-related vulnerabilities. These include use of repeated keys, use of RSA keys that were factorable due to repeated primes, and use of DSA keys that were compromised by repeated signature randomness. Under the theory that vulnerable repeated keys were generated by embedded or headless devices with defective designs, we also report the number of hosts that we identified as these device models. Many of these hosts may be at risk even though we did not specifically observe repeats of their keys.

Aug 9, 2021

FUNDAMENTAL FLAW IN RNGS AFFECTS MANY IOT DEVICES

By Dennis Fisher

https://duo.com/decipher/fundamental-flaw-in-rngs-affects-many-iot-devices

Seeding with Time is BAD

```
Random gen = new Random();
gen.setSeed(
  new Date().getTime()
);
Int val = gen.nextInt();
```

Brute Force Attacks

Brute-force attacks consist of trying *all* possibilities until the correct one is found.

- Rarely the most efficient
- Usually trivially scalable via concurrency
- 100% successful given enough time

- Defense: Make “enough time” infeasible
  - Order of “heat death of the universe”

Seeding with Time is BAD

```
Random gen = new Random();
gen.setSeed(
  new Date().getTime()
);
Int val = gen.nextInt();
```

- Computers tell time since the “epoch”
  - 01Jan1970 @ 00:00
  - Increments of seconds, ms, or ns
- 1 year =~= 34B ms
  - Approx 2^{35} ms

< 1 hour to try all w/ a VERY naïve implementation

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

Properties of Secure Channel

Is a secure source of randomness *sufficient* for achieving a secure channel?

- Confidentiality
- Message Integrity
- Sender Authenticity

No. But is necessary for achieving a secure channel.

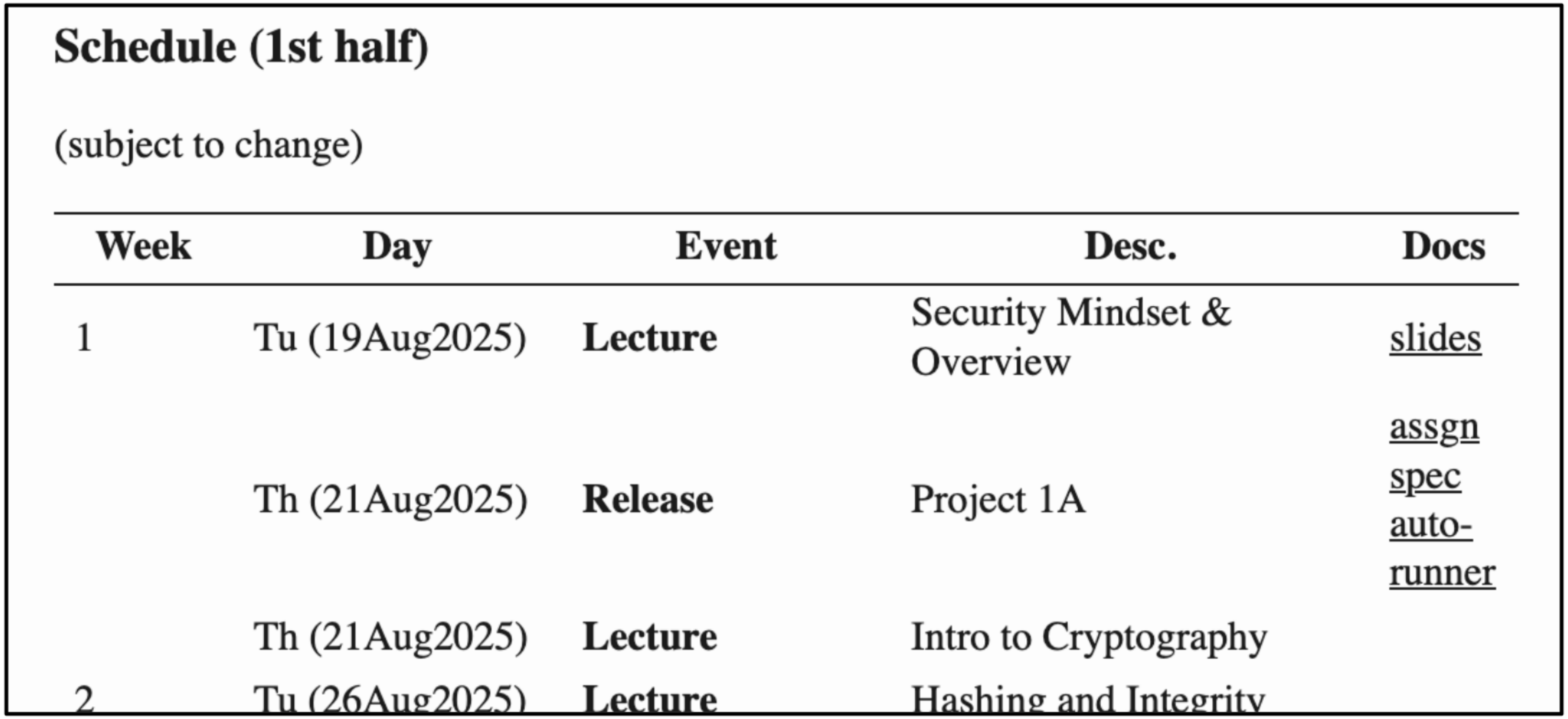
Computer and Network Security

Lecture 02:
Intro to Cryptography

COMP-5370/6370
Fall 2025

Course Notes

- Project 1A is live and due in two weeks



Schedule (1st half)

(subject to change)

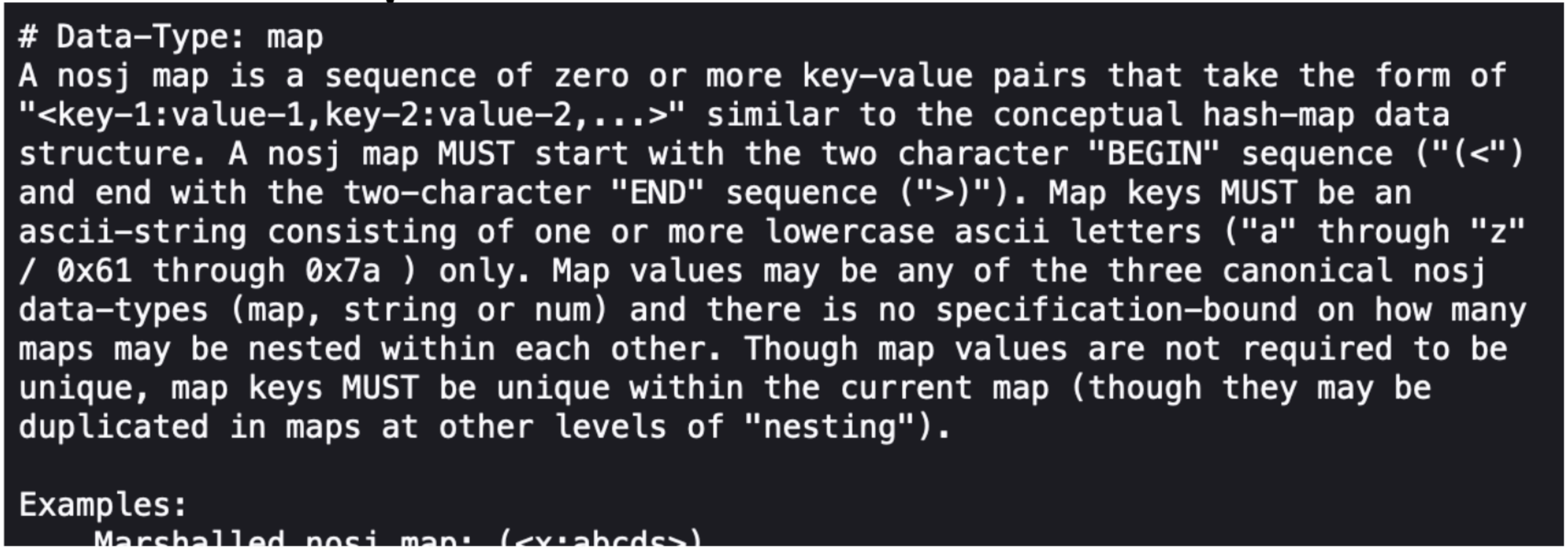
Week	Day	Event	Desc.	Docs
1	Tu (19Aug2025)	**Lecture**	Security Mindset & Overview	slides
	Th (21Aug2025)	**Release**	Project 1A	assgn spec auto-runner
	Th (21Aug2025)	**Lecture**	Intro to Cryptography	
2	Tu (26Aug2025)	**Lecture**	Hashing and Integrity	

Project 1A

Input: `(<abc:defs>)`

Project 1A

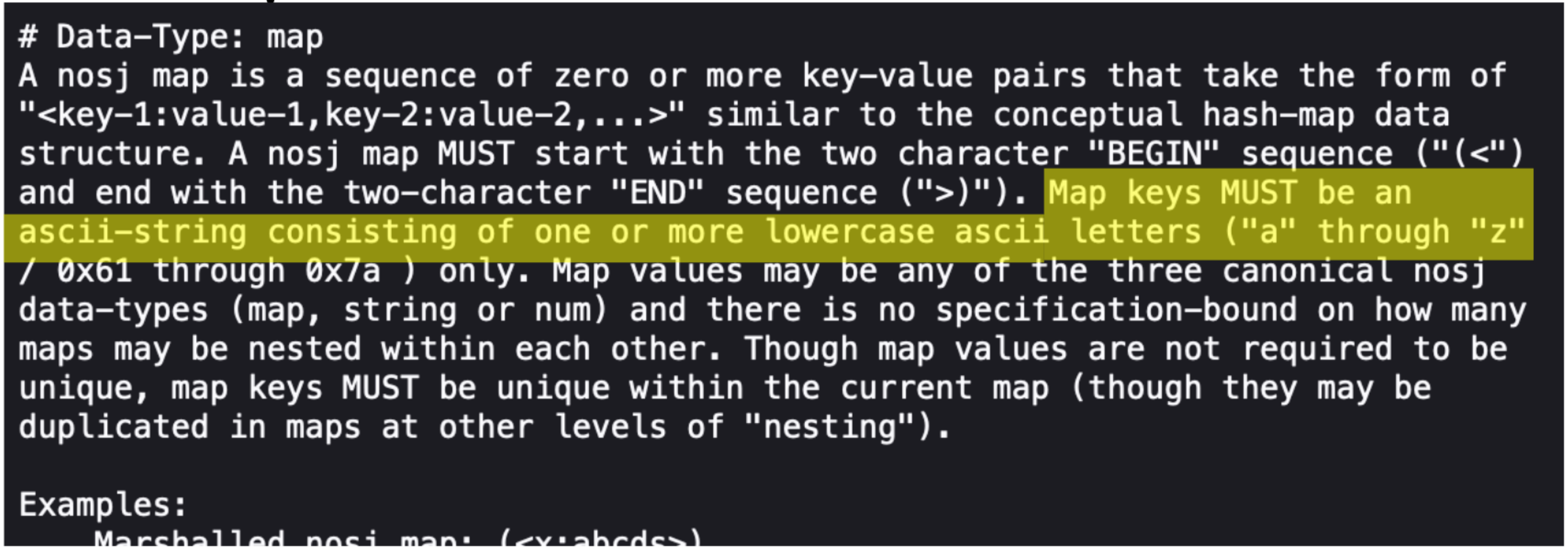
Input: `(<abc:defs>)`



```
# 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

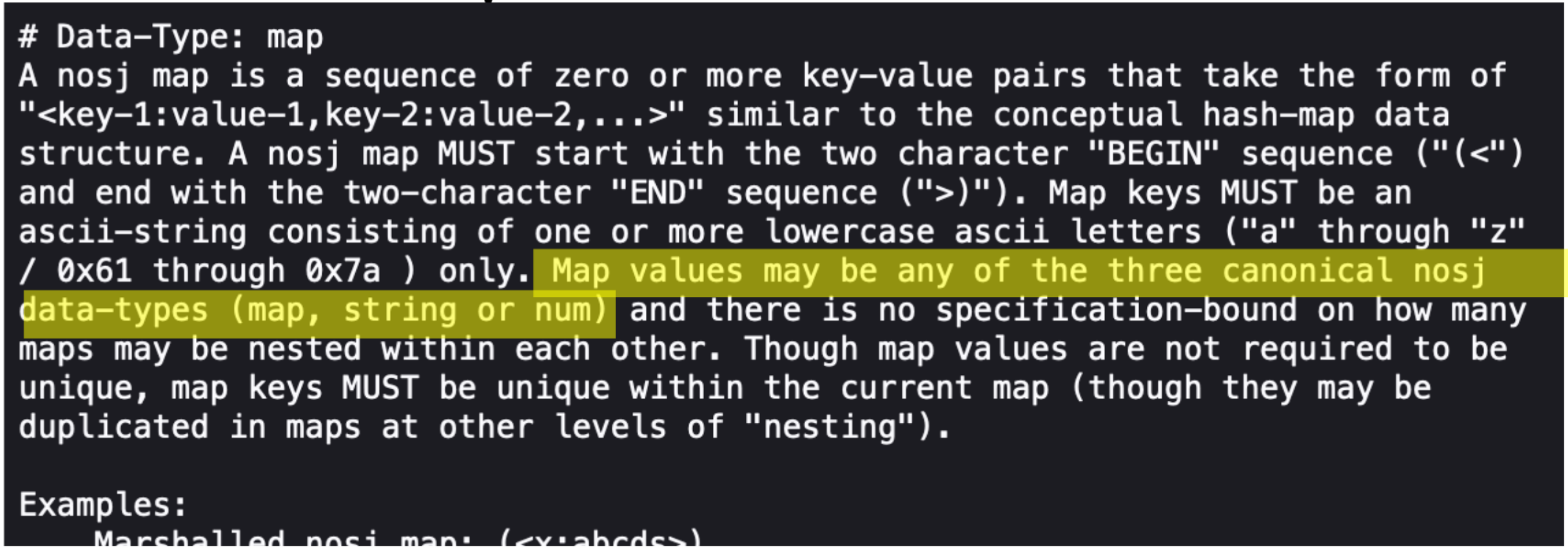
Input: `(<abc:defs>)`   Key: `“‘abc’”`



```
# 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

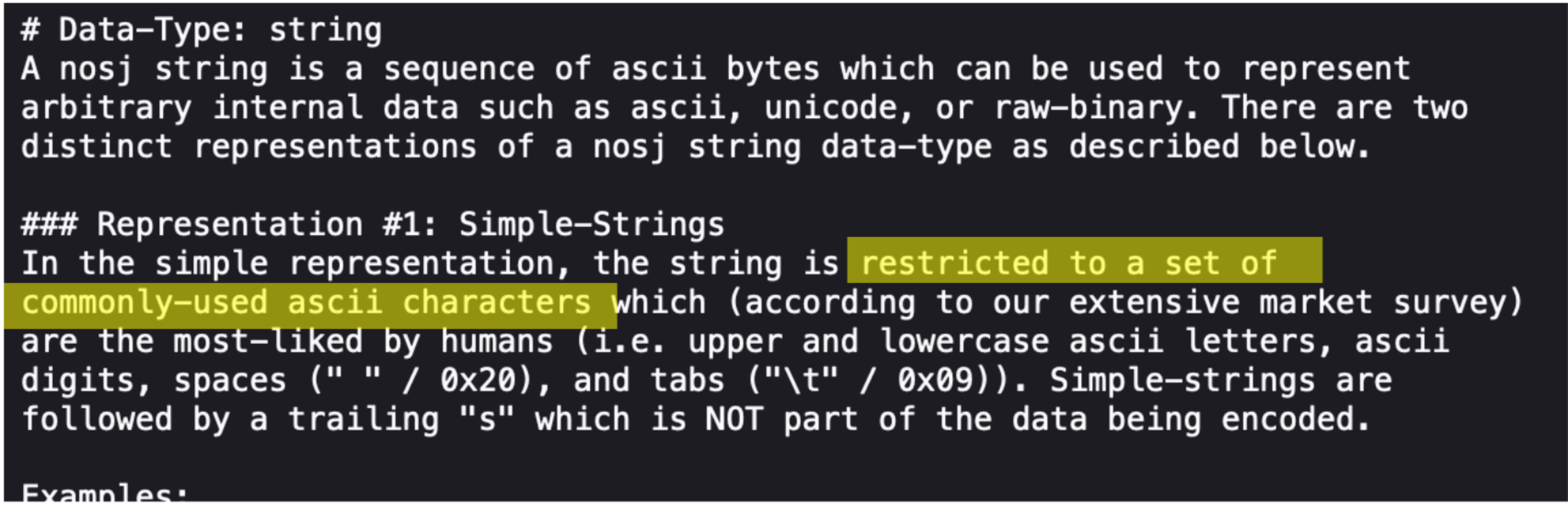
Input: (<abc:defs>) Key: ‘‘abc’’



```
# 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’’`



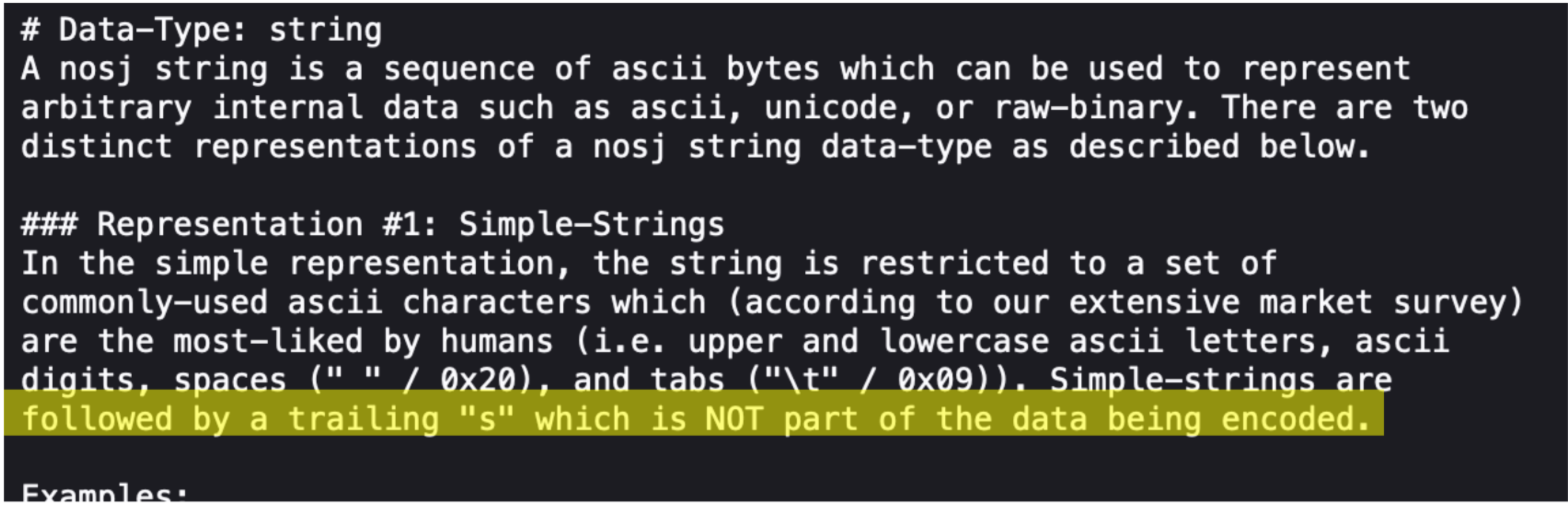
```
# Data-Type: string
A nosj string is a sequence of ascii bytes which can be used to represent
arbitrary internal data such as ascii, unicode, or raw-binary. There are two
distinct representations of a nosj string data-type as described below.

### 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>)

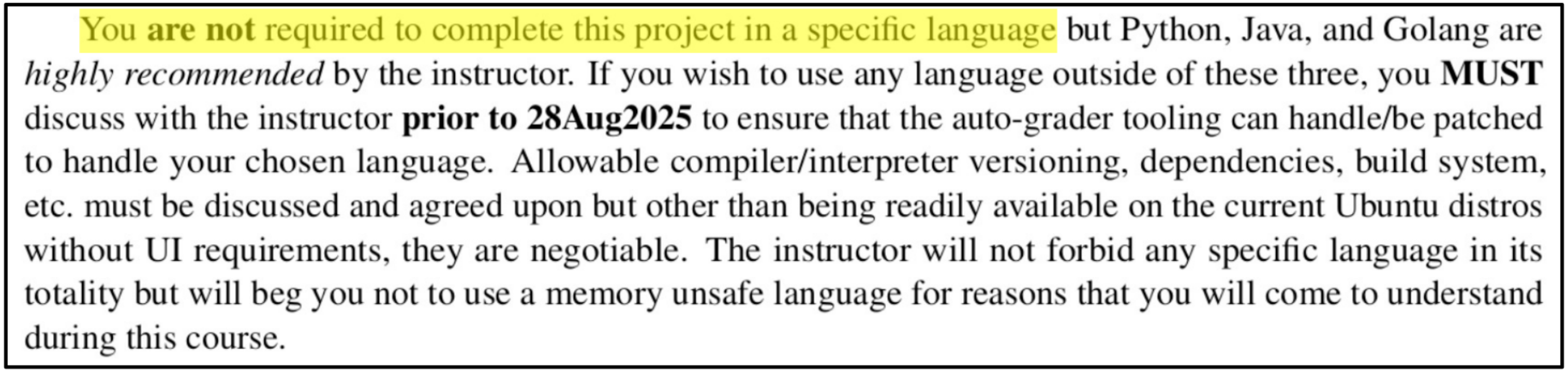
Key: ‘‘abc’’

Value: ‘‘def’’



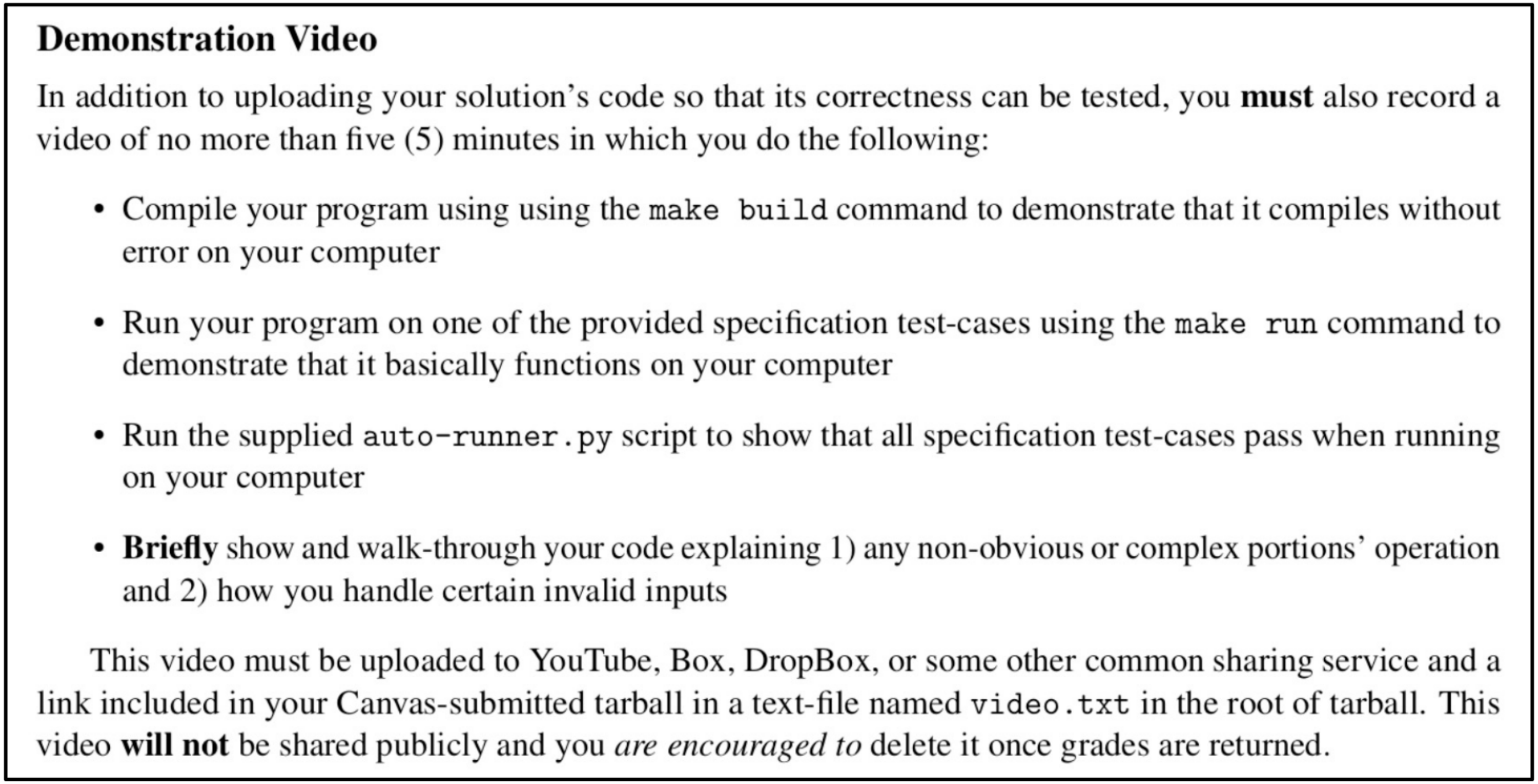
```
# Data-Type: string
A nosj string is a sequence of ascii bytes which can be used to represent
arbitrary internal data such as ascii, unicode, or raw-binary. There are two
distinct representations of a nosj string data-type as described below.

### 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



You **are not** required to complete this project in a specific language but Python, Java, and Golang are *highly recommended* by the instructor. If you wish to use any language outside of these three, you **MUST** discuss with the instructor **prior to 28Aug2025** to ensure that the auto-grader tooling can handle/be patched to handle your chosen language. Allowable compiler/interpreter versioning, dependencies, build system, etc. must be discussed and agreed upon but other than being readily available on the current Ubuntu distros without UI requirements, they are negotiable. The instructor will not forbid any specific language in its totality but will beg you not to use a memory unsafe language for reasons that you will come to understand during this course.

Demo Video



Demonstration Video

In addition to uploading your solution's code so that its correctness can be tested, you **must** also record a video of no more than five (5) minutes in which you do the following:

- Compile your program using using the `make build` command to demonstrate that it compiles without error on your computer
- Run your program on one of the provided specification test-cases using the `make run` command to demonstrate that it basically functions on your computer
- Run the supplied `auto-runner.py` script to show that all specification test-cases pass when running on your computer
- **Briefly** show and walk-through your code explaining 1) any non-obvious or complex portions' operation and 2) how you handle certain invalid inputs

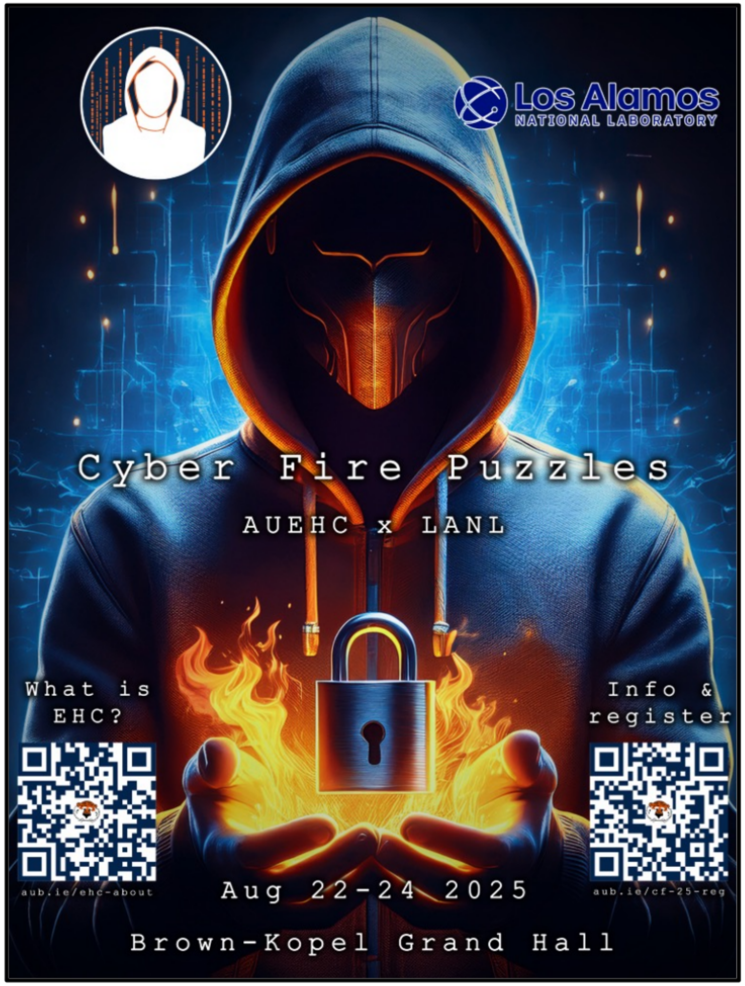
This video must be uploaded to YouTube, Box, DropBox, or some other common sharing service and a link included in your Canvas-submitted tarball in a text-file named `video.txt` in the root of tarball. This video **will not** be shared publicly and you *are encouraged to* delete it once grades are returned.

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**

CTF This Weekend



- “Capture the Flag” challenges
- Register via link in your email
  - $30 registration but meals + snacks/drinks provided

Computer and Network Security

Lecture 02:
Intro to Cryptography

COMP-5370/6370
Fall 2025