Computer and Network Security

Lecture 07:
Sender Authenticity

COMP-5370/6370
Fall 2024

WARNING

I AM NOT A CRYPTOGRAPHER

WARNING

YOU ARE NOT A CRYPTOGRAPHER

- 4th Rule: Don’t roll your own crypto
- 5th Rule: Don’t roll your own crypto
- 6th Rule: Don’t roll your own crypto
- 7th Rule: Don’t roll your own crypto

Public Key Cryptography

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

Not a single
shared secret
between
all parties

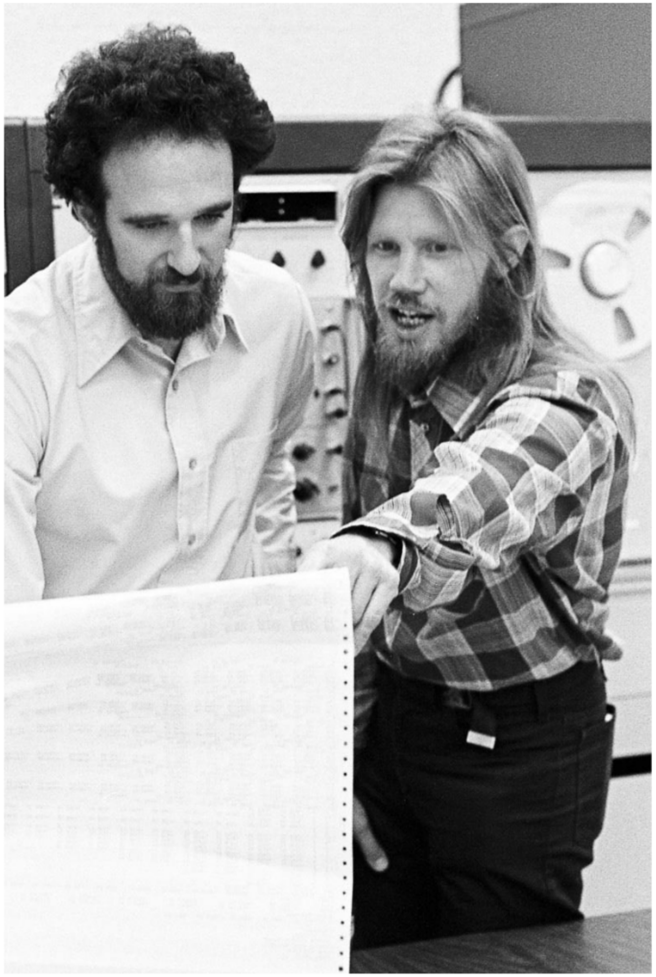
Public key
&
Private key

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

Using PubKey Crypto

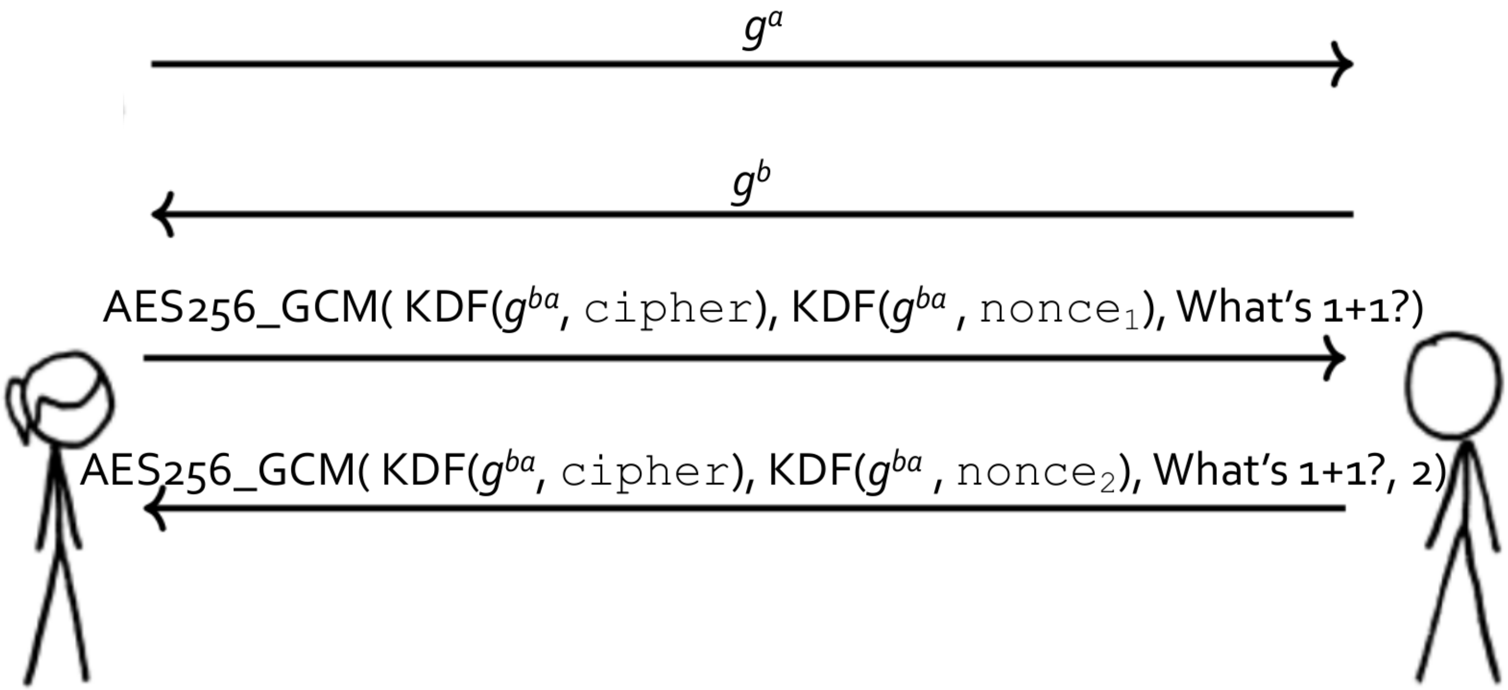
- Key Exchange
  - Create a shared secret in the presence of a passive attacker (Eve)

Diffie-Hellman Key Exchange



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

Building a Secure Channel



g^a

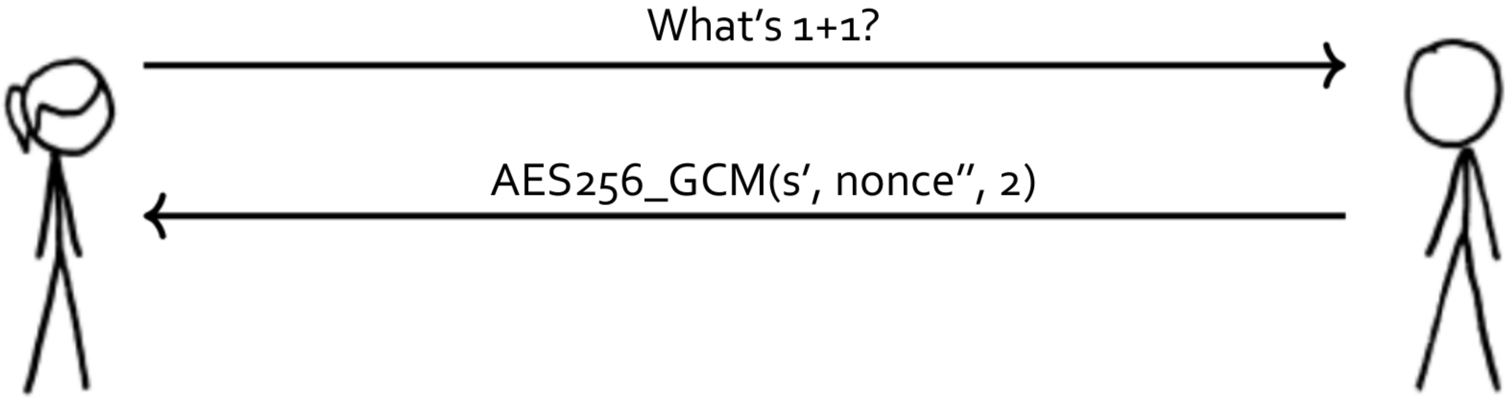
g^b

AES256_GCM(KDF(g^{ba}, `cipher`), KDF(g^{ba} , $\texttt{nonce}_1$), What's 1+1?)

AES256_GCM(KDF(g^{ba}, `cipher`), KDF(g^{ba} , $\texttt{nonce}_2$), What's 1+1?, 2)

Building a Secure Channel

- 🙂 **Confidentiality**
- 🙂 **Message Integrity**
- ☹ **Sender Authenticity**

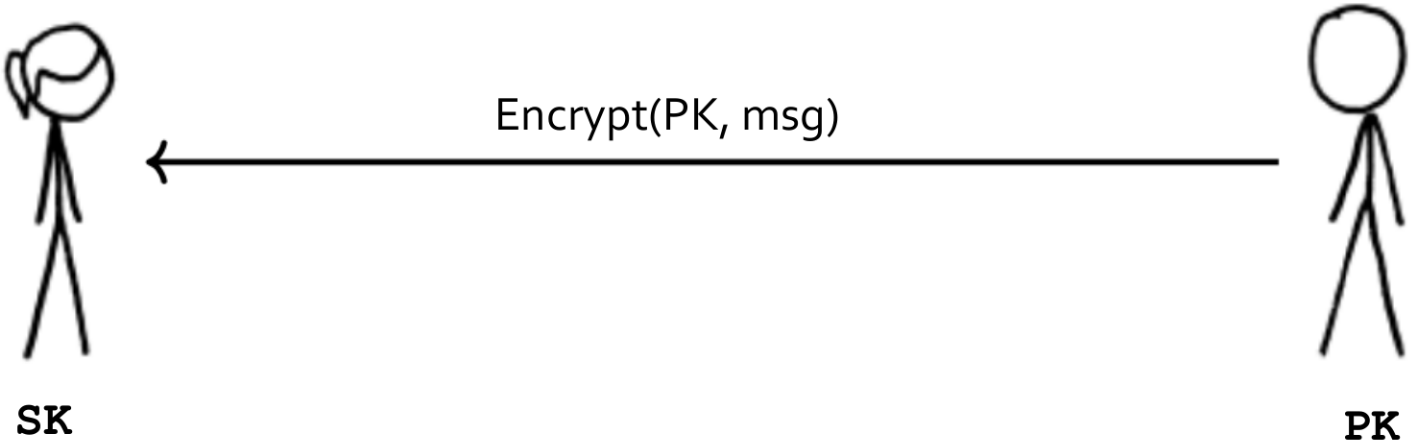


Using PubKey Crypto

- Key Exchange
  - Create a shared secret in the presence of a passive attacker (Eve)
- Encryption/Decryption
  - Encrypt w/ **public key**, decrypt w/ **private key**
  - Allows anyone to **send** information securely as long as have public key

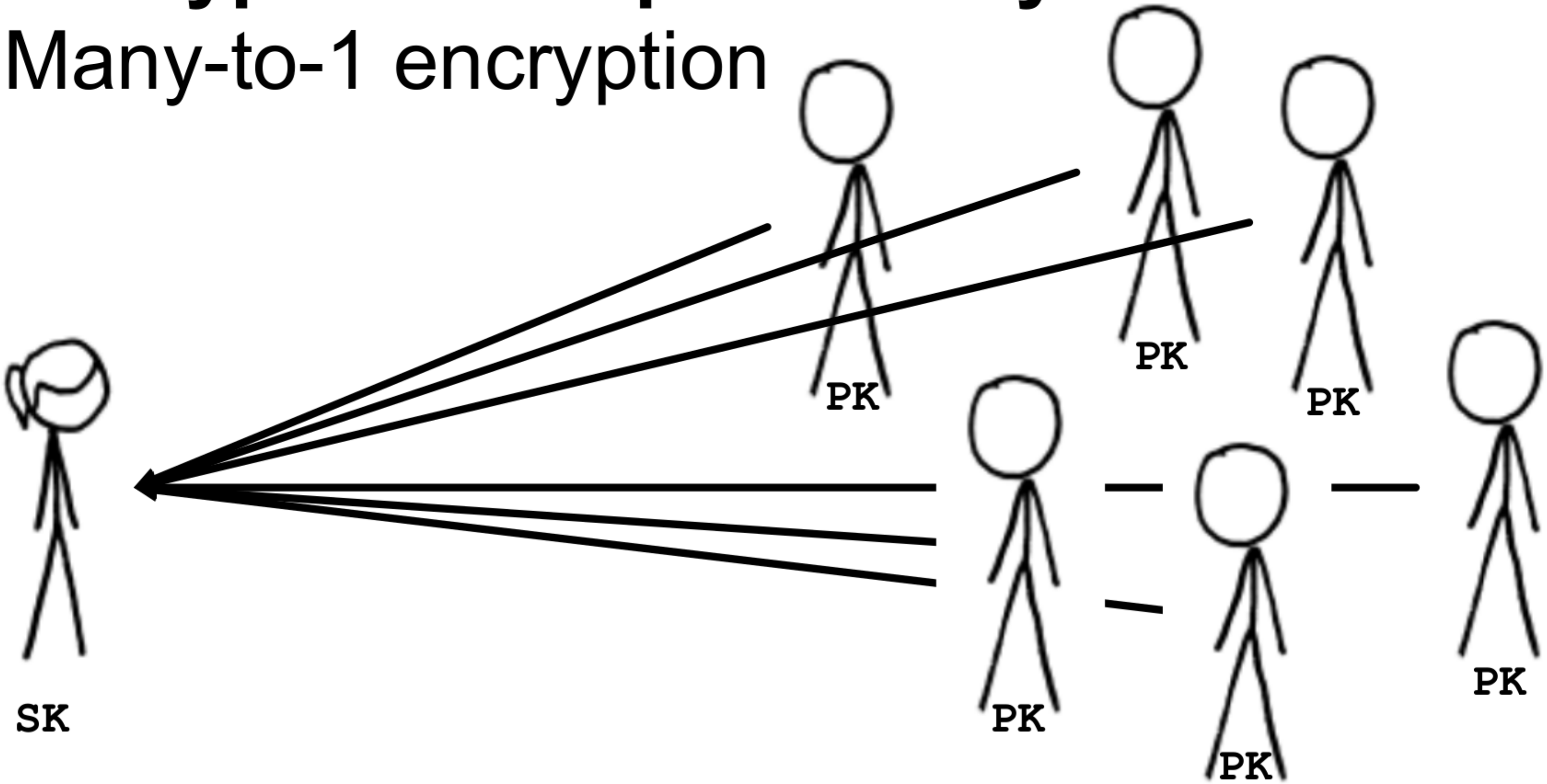
PubKey Encrypt/Decrypt

- **Encrypt** with the **public key**
- **Decrypt** with the **private key**



PubKey Encrypt/Decrypt

- **Encrypt** with the **public key**
- **Decrypt** with the **private key**
- Many-to-1 encryption



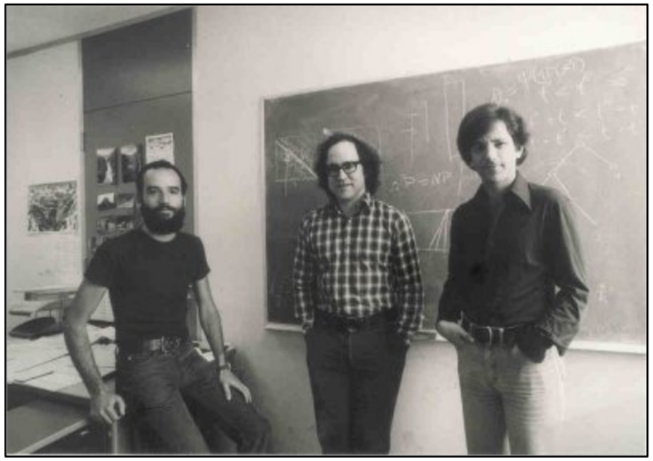
Digital Signature

A **digital signature** is a cryptographic value that allows anyone to *verify* data's source.

- Create a signature (`sig`) with **private key**
- Validate `sig` with **public key**

- Signature schemes ***hash*** a the message as part of the operation

RSA Cryptosystem



- 1978 – Ron **R**ivest, Adi **S**hamir, & Leonard **A**dleman

- Modular Exponentiation w/ non-prime modulus
  - *If you multiply a value by itself enough times over a finite field … it eventually returns to its original value.*

RSA Cryptosystem

- 1973 – Clifford Cocks
  - GCHQ cryptographer
  - Classified until 1997
- Modular Exponentiation w/ non-prime modulus
  - *If you multiply a value by itself enough times over a finite field … it eventually becomes a cycle and returns to its original value*

RSA Key Generation

- Generate two large primes *p* and *q*
  - *p* and *q*
- Calculate modulus *n* from *p* and *q*
  - $n = p * q$ (|*n*| is the RSA "length")
- Select relatively prime public exponent *e*
  - Usually 3 or 65,537
- Find a private exponent *d*
  - $(e * d) \bmod lcm((p-1) * (q-1)) = 1$
- Priv-Key = (*d*, *n*)
- Pub-Key = (*e*, *n*)

Textbook RSA Enc. & Dec.

If you multiply a value by itself enough times over a finite field … it eventually becomes a cycle and returns to its original value.

- Encrypt with the public key *(e, n)*
  - $CT = PT^e \bmod n$
- Decrypt with the private key *(d, n)*
  - $PT = CT^d \bmod n$

Textbook RSA Sign/Verify

If you multiply a value by itself enough times over a finite field … it eventually becomes a cycle and returns to its original value.

- Sign with the private key *(d, n)*
  - $\mathtt{sig} = (\mathtt{hash}(\mathtt{data}))^{\mathtt{d}} \ \mathtt{mod}\ \mathtt{n}$
- Verify with the public key *(e, n)*
  - $\mathtt{sig}^{\mathtt{e}} \ \mathtt{mod}\ \mathtt{n}\ \mathtt{=?=}\ \mathtt{hash}(\mathtt{data})$

Security of RSA

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

Integer Factorization Problem	**RSA Problem**
Given *n* it is hard to recover *p* and *q*	Given only the pub-key, it is hard to perform a priv-key operation

Safe RSA Parameters

- Correctly generated 2048-bit modulus
  - Thought to be safe
  - Widely used in the real-world
- Correctly generated 3072-bit modulus
  - Thought to be safe
  - Relatively rare in the real-world
  - CNSA approved

Canonical RSA Vulnerabilities

- Brute-force computation overmatch
  - Can factor 512-bit n on EC2 for ~$75
  - Assumed $100M of ASICs for 1024b modulus
- Poor randomness when selecting p and q
- Insecure strategy for generating p and q
  - Vulnerable example: $p = \text{prime}_n$, $q = \text{prime}_{n+1}$
- Algorithmic advances
  - Pre-Quantum: Number Field Sieve (NFS)
  - Post-Quantum: Shor’s algorithm

Using PubKey Crypto

- Key Exchange
  - Create a shared secret in the presence of a passive attacker (Eve)
- Encryption/Decryption
  - Encrypt w/ **public key**, decrypt w/ **private key**
  - Allows anyone to **send** information securely as long as have public key
- Digital Signatures
  - Sign w/ **private key**, verify w/ **public key**
  - Allows anyone to **receive** data w/ known-origin

Debating Rolling Own Crypto?

RSA CIPHERTEXT IS MALLEABLE

$$CT_{new} = CT_1 * CT_2 \bmod n$$

$$PT_{new} = (CT_{new})^d \bmod n$$

$$CT_{new} = Encrypt(PT_1 * PT_2$$

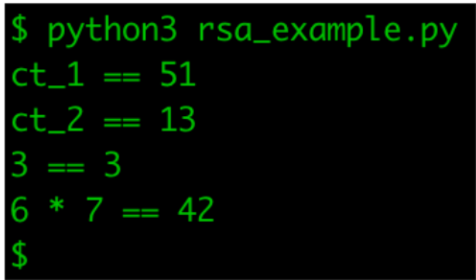
```
p = 5
q = 11
n = p*q
e = 3
d = 27

def encrypt(pt): return pow(pt, e, n)
def decrypt(ct): return pow(ct, d, n)

ct_1 = encrypt(6)
ct_2 = encrypt(7)
print('ct_1 ==', ct_1)
print('ct_2 ==', ct_2)

ct_new = (ct_1 * ct_2) % n
print(encrypt(42), '==', ct_new)

print('6 * 7 ==', decrypt(ct_new))
assert(decrypt(encrypt(42)) == decrypt(ct_new))
```



```
$ python3 rsa_example.py
ct_1 == 51
ct_2 == 13
3 == 3
6 * 7 == 42
$
```

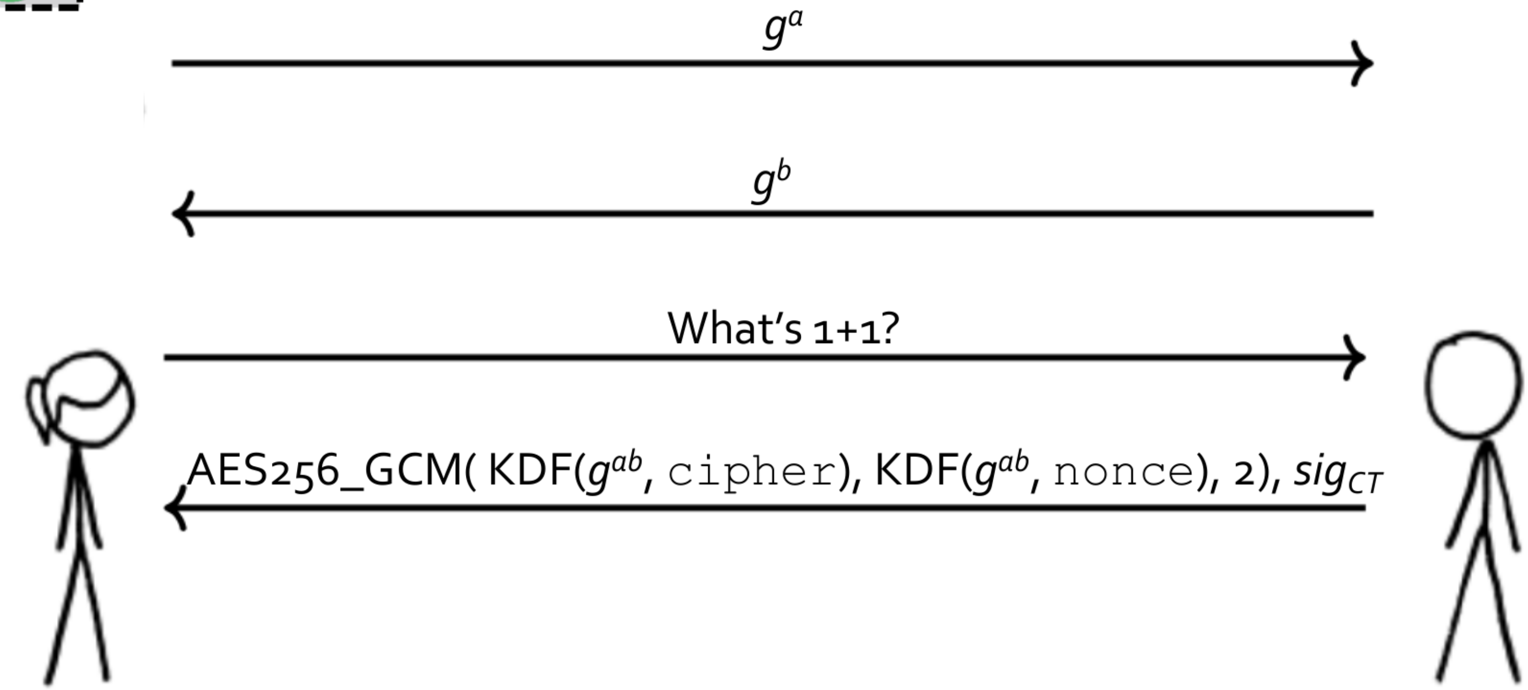
There are many, many more

Building a Secure Channel

☺ **Confidentiality**

☺ **Message Integrity**

☺ **Sender Authenticity**



g^a

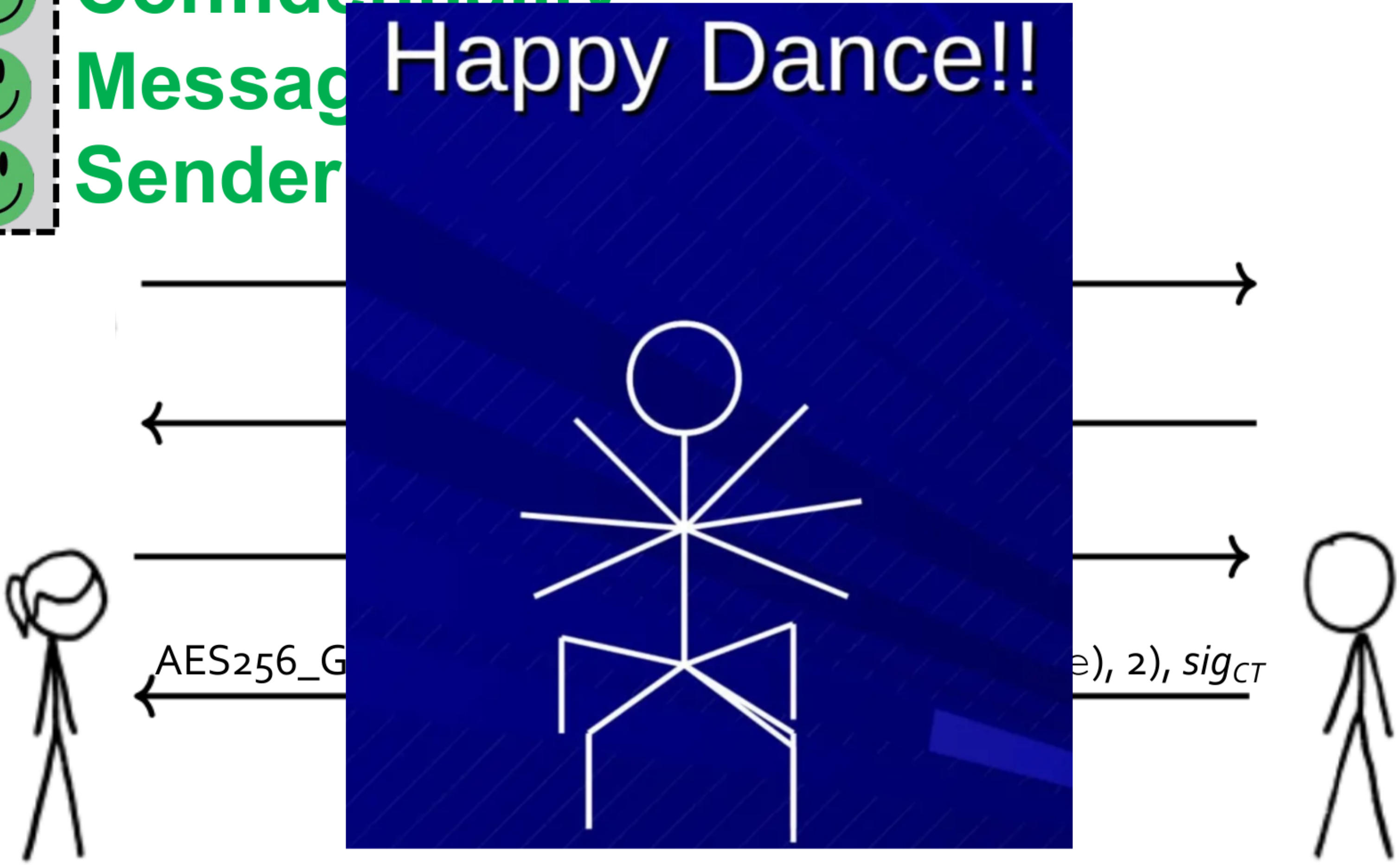
g^b

What's 1+1?

AES256_GCM(KDF(g^{ab}, `cipher`), KDF(g^{ab}, `nonce`), 2), sig_{CT}

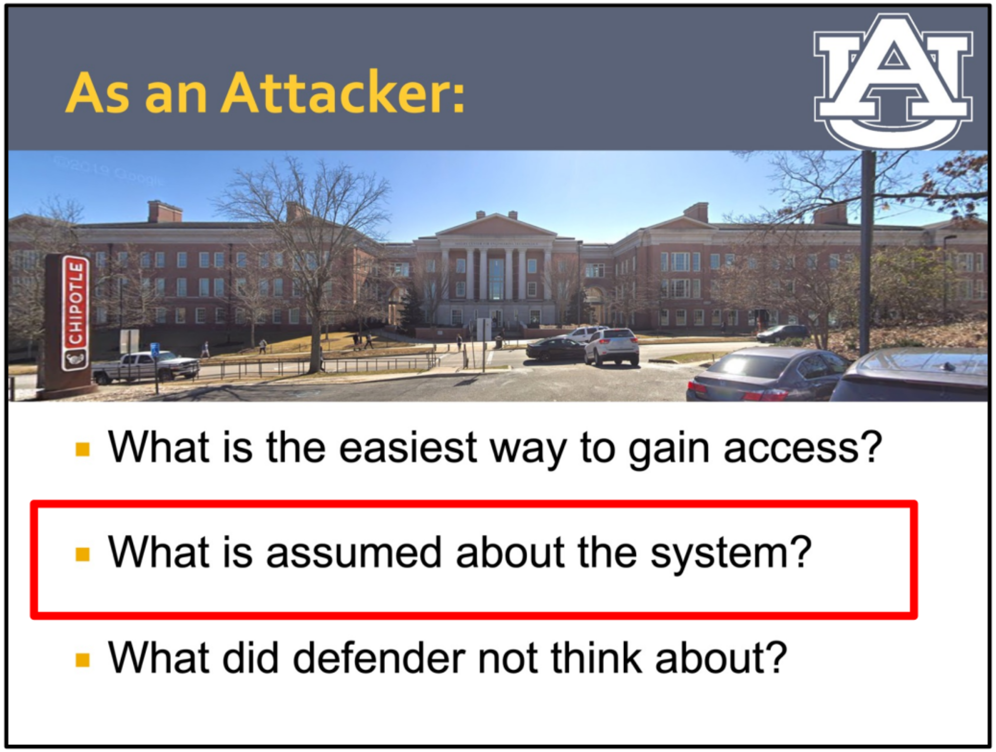
Building a Secure Channel

- ☺ Confidentiality
- ☺ Messag
- ☺ Sender



Security Analysis

In order to use cryptography ***safely***, you have to account for *that the use-case is*.



As an Attacker:

- What is the easiest way to gain access?
- What is assumed about the system?
- What did defender not think about?

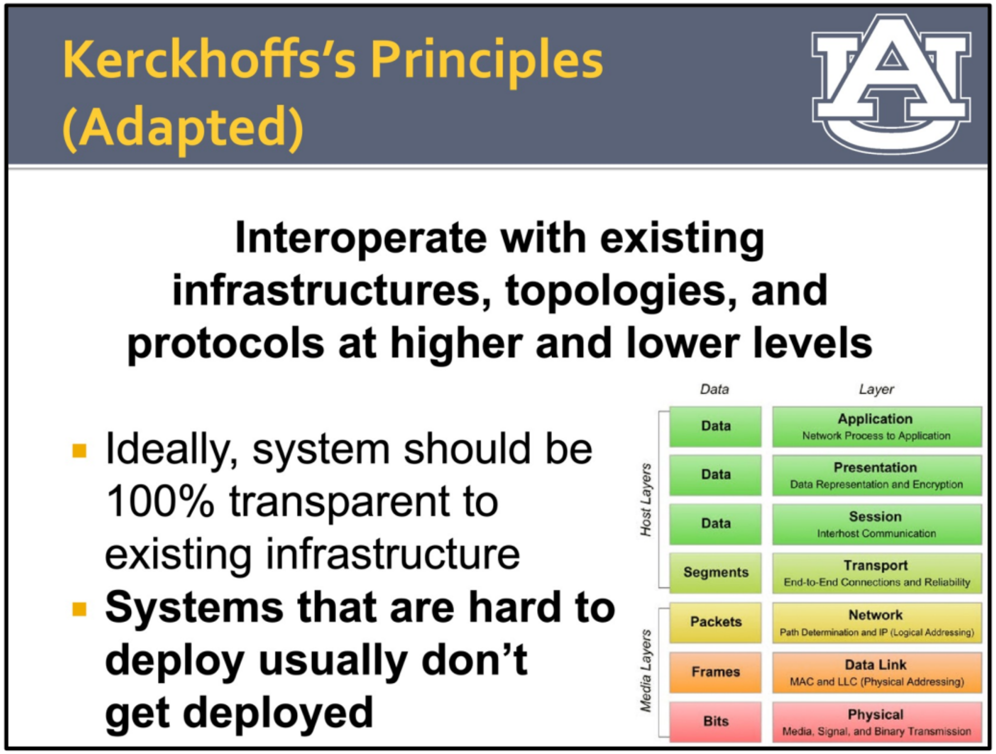
Security Analysis

What assumptions have we (the designers) made in order to build our Secure Channel?

- An insecure channel exists/can be leveraged
- There are *two* actors involved (Alice/Bob)
- Alice and Bob's interaction is "online"
- Alice and Bob have each others' pub-keys
- All our crypto primitives are safe to use

Security Analysis

In order to build technology ***in a helpful way***, you have to plan for *deployment/usage.*



Kerckhoffs's Principles (Adapted)

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

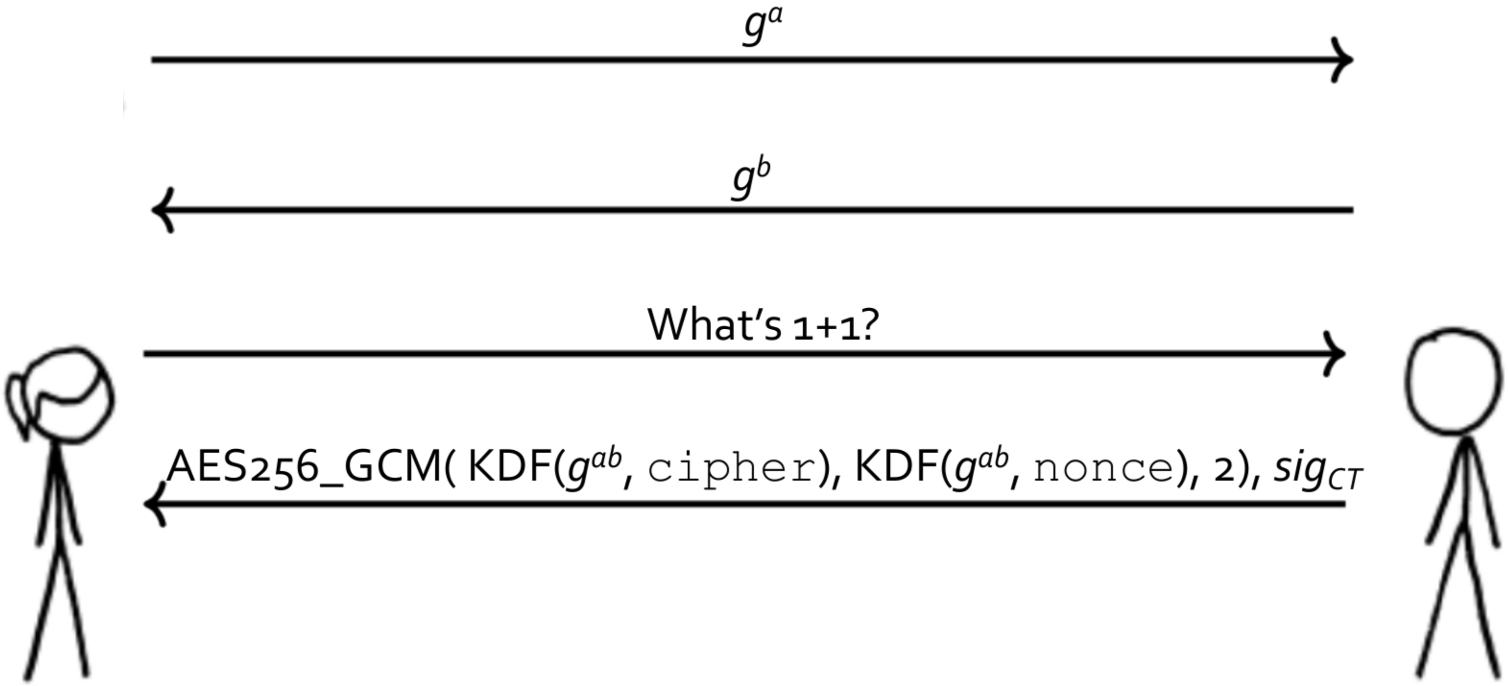
- Ideally, system should be 100% transparent to existing infrastructure
- **Systems that are hard to deploy usually don't get deployed**

	Data	*Layer*
Host Layers	**Data**	**Application** Network Process to Application
	Data	**Presentation** Data Representation and Encryption
	Data	**Session** Interhost Communication
	Segments	**Transport** End-to-End Connections and Reliability
Media Layers	**Packets**	**Network** Path Determination and IP (Logical Addressing)
	Frames	**Data Link** MAC and LLC (Physical Addressing)
	Bits	**Physical** Media, Signal, and Binary Transmission

Security Analysis

Alice & Bob have a secure channel to talk.

Is this channel useable in the real-world?



g^a

g^b

What's 1+1?

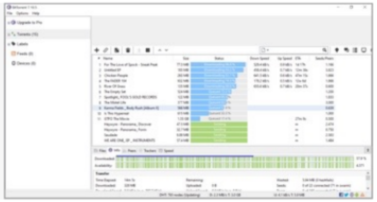
AES256_GCM(KDF(g^{ab}, `cipher`), KDF(g^{ab}, `nonce`), 2), sig_{CT}

Security Analysis

- Relatively poor performance
  - High per-message computation and bandwidth
- No backwards compatibility
  - Very difficult to update or interoperate
- No protocol flexibility
  - Everything (cipher, hash, etc primitive) is ***fixed***
- Frequent use of long-term keys

But What About …

- What if Alice only wants to talk to herself?
  - EX: Encrypting backup
- What if Alice only cares about Sender Authenticity + Integrity?
  - EX: “Bob said XXXX and everyone should know”
- What if Alice only cares about provenance of static data?
  - EX: Peer-to-Peer file sharing



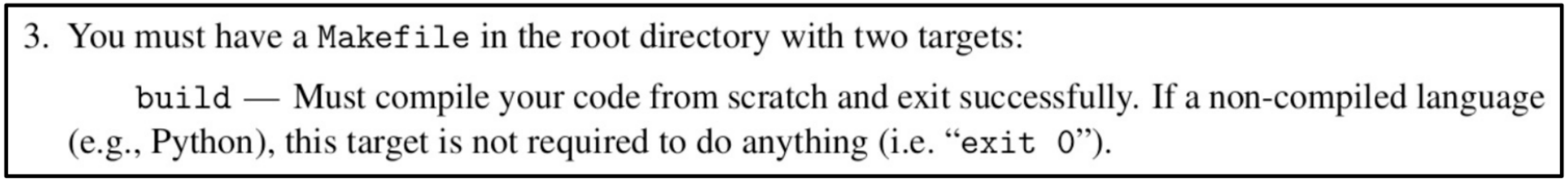
Computer and Network Security

Lecture 07:
Sender Authenticity

COMP-5370/6370
Fall 2024

Project 1-A

We have to be able to build your code.

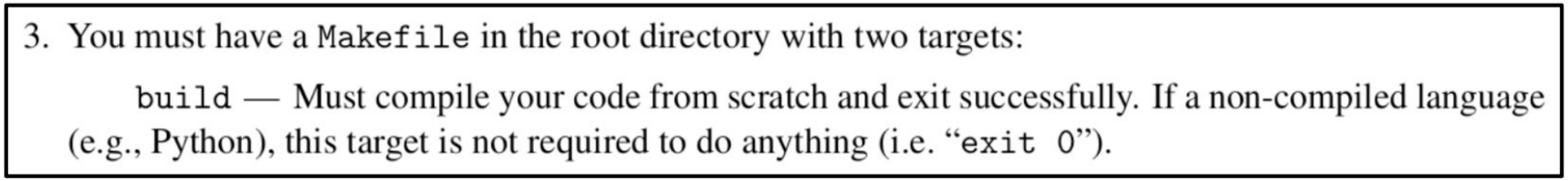


3. You must have a `Makefile` in the root directory with two targets:

   `build` — Must compile your code from scratch and exit successfully. If a non-compiled language (e.g., Python), this target is not required to do anything (i.e. “`exit 0`”).

Project 1-A

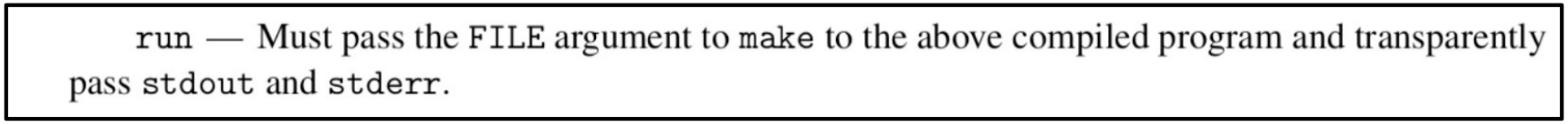
We have to be able to build your code.



3. You must have a `Makefile` in the root directory with two targets:

   `build` — Must compile your code from scratch and exit successfully. If a non-compiled language (e.g., Python), this target is not required to do anything (i.e. "`exit 0`").

We have to be able to build your code.

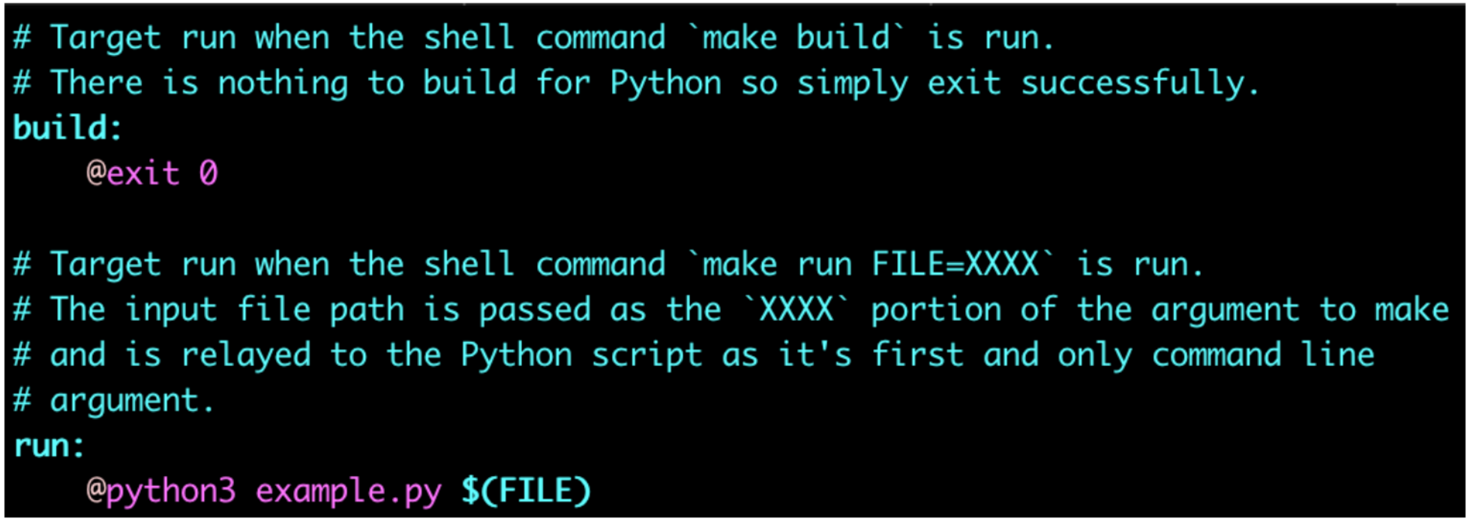


`run` — Must pass the `FILE` argument to `make` to the above compiled program and transparently pass `stdout` and `stderr`.

Regrade Requests

- Regrade requests come in 2 forms:
    - Fix things that break the auto-grader
    - Fix *minor* things that cause major deduction

Project 1-A



```
# Target run when the shell command `make build` is run.
# There is nothing to build for Python so simply exit successfully.
build:
    @exit 0

# Target run when the shell command `make run FILE=XXXX` is run.
# The input file path is passed as the `XXXX` portion of the argument to make
# and is relayed to the Python script as it's first and only command line
# argument.
run:
    @python3 example.py $(FILE)
```

This is important!

Auto-Grader Errors

- Removed the “@” in Makefile and have extra line of output to stdout?
  - You can add an “@” sign.
- Submission structure wrong?
  - You can move your files around
- Makefile isn’t a makefile?
  - You can fix your Makefile
- Etc.

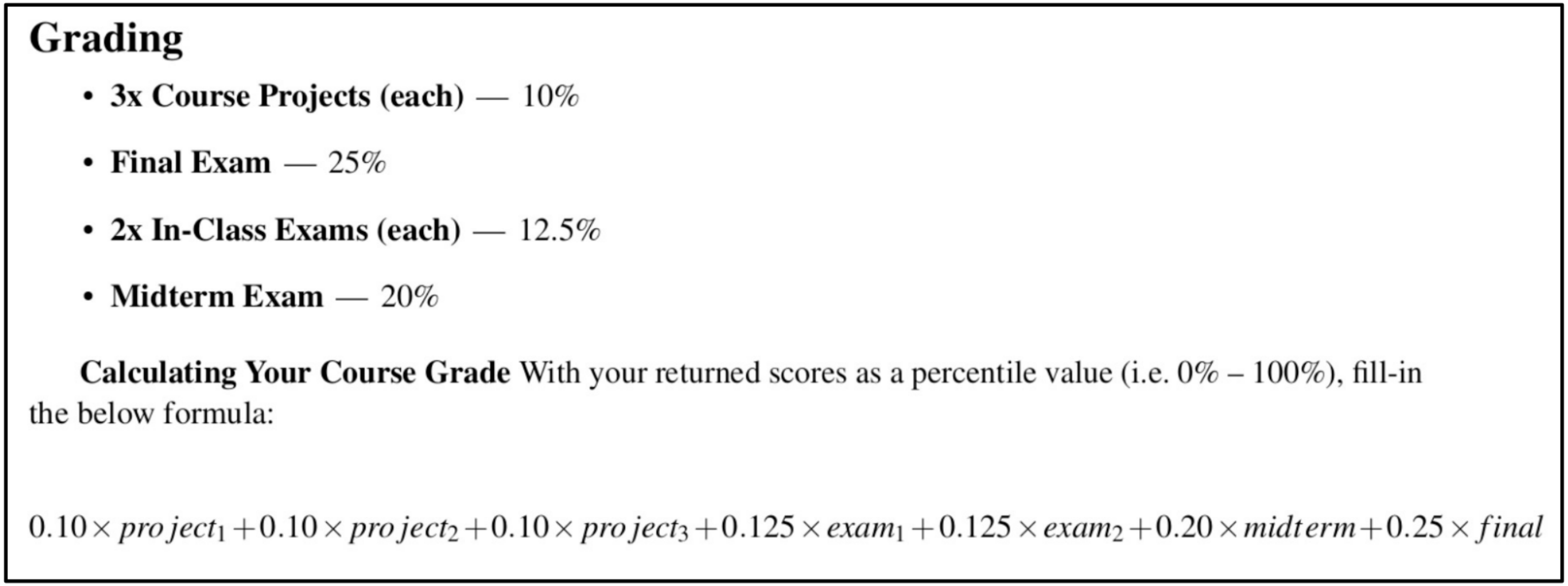
YOU DON’T GET AN EXTRA WEEK
TO FINISH IMPLEMENTING

Fundamental Misunderstandings

- Print to the wrong handle (stdout vs. stderr)?
  - You can change to the correct handle
- Print “`--- simple-string ---` ” and not “`--- string ---`”?
  - You can change your print statement’s literal
- Etc.

YOU DON’T GET AN EXTRA WEEK TO FINISH IMPLEMENTING

Let’s do some Math



Grading

- **3x Course Projects (each)** — 10%
- **Final Exam** — 25%
- **2x In-Class Exams (each)** — 12.5%
- **Midterm Exam** — 20%

Calculating Your Course Grade With your returned scores as a percentile value (i.e. 0% – 100%), fill-in the below formula:

$$0.10 \times project_1 + 0.10 \times project_2 + 0.10 \times project_3 + 0.125 \times exam_1 + 0.125 \times exam_2 + 0.20 \times midterm + 0.25 \times final$$

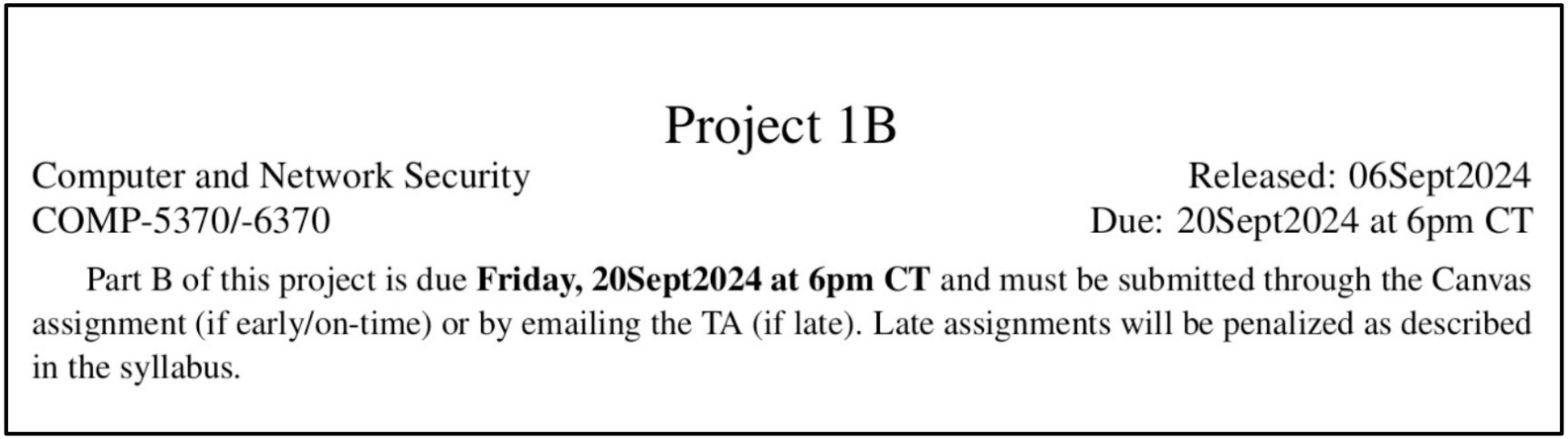
- A zero (0) on Project 1A means that you have a max final grade of 95% (A)

Regrade Requests

- Regrade requests come in 2 forms:
  - Fix things that break the auto-grader
  - Fix *minor* things that cause major deduction
- **Each rejection costs you points**
- The “clock” does not restart every time you send us a new version.

Project 1-B

- Released Friday
- Due next Friday, 20Sept2024



Project 1B

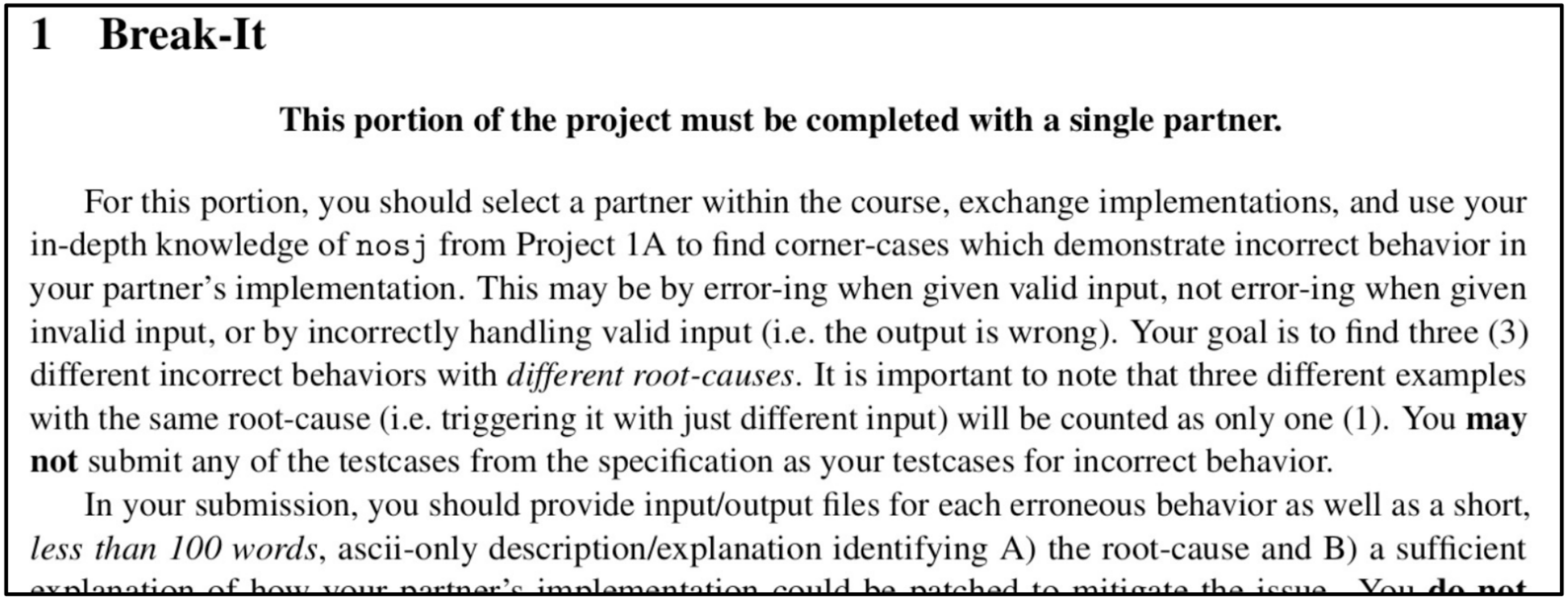
Computer and Network Security
COMP-5370/-6370

Released: 06Sept2024
Due: 20Sept2024 at 6pm CT

Part B of this project is due **Friday, 20Sept2024 at 6pm CT** and must be submitted through the Canvas assignment (if early/on-time) or by emailing the TA (if late). Late assignments will be penalized as described in the syllabus.

Project 1-B --- Part 1

- Trade implementations with a partner and find/demonstrate incorrect behavior
  - Must have **different root-causes**



1 Break-It

This portion of the project must be completed with a single partner.

For this portion, you should select a partner within the course, exchange implementations, and use your in-depth knowledge of `nosj` from Project 1A to find corner-cases which demonstrate incorrect behavior in your partner's implementation. This may be by error-ing when given valid input, not error-ing when given invalid input, or by incorrectly handling valid input (i.e. the output is wrong). Your goal is to find three (3) different incorrect behaviors with *different root-causes*. It is important to note that three different examples with the same root-cause (i.e. triggering it with just different input) will be counted as only one (1). You **may not** submit any of the testcases from the specification as your testcases for incorrect behavior.

In your submission, you should provide input/output files for each erroneous behavior as well as a short, *less than 100 words*, ascii-only description/explanation identifying A) the root-cause and B) a sufficient explanation of how your partner's implementation could be patched to mitigate the issue. You **do not**

Project 1-B --- Part 1

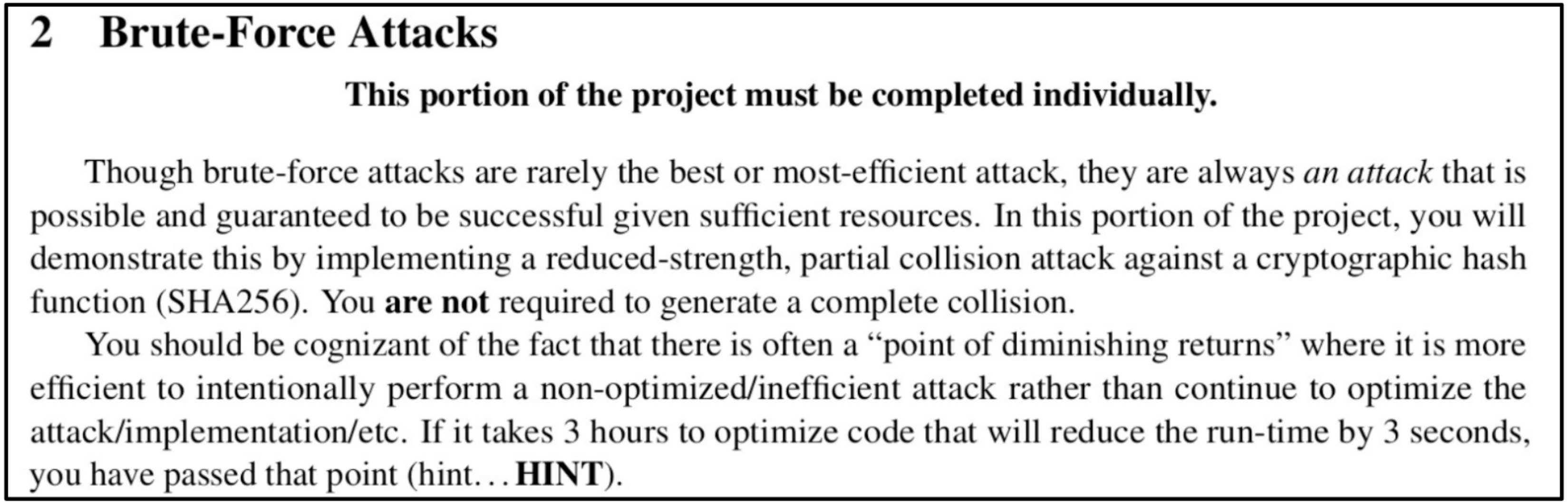
- You may only have 1 Partner
  - Your 1A implementation must be used by only one person for 1B

- If you can’t find a partner by ***this*** Friday, let us know and we’ll work it out

YOU SHOULD NOT BE WAITING ON YOUR PARTNER’S IMPLEMENTATION

Project 1-B --- Part 2

- Generate a partial SHA256 hash collision via brute-force attack



2 Brute-Force Attacks

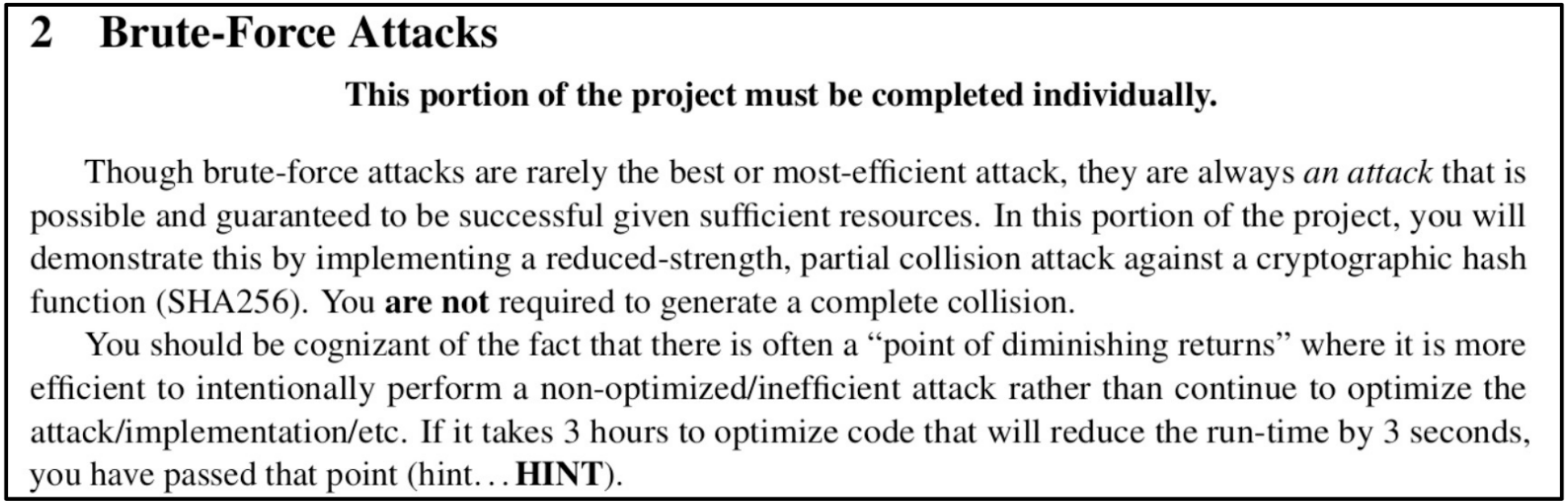
This portion of the project must be completed individually.

Though brute-force attacks are rarely the best or most-efficient attack, they are always *an attack* that is possible and guaranteed to be successful given sufficient resources. In this portion of the project, you will demonstrate this by implementing a reduced-strength, partial collision attack against a cryptographic hash function (SHA256). You **are not** required to generate a complete collision.

You should be cognizant of the fact that there is often a "point of diminishing returns" where it is more efficient to intentionally perform a non-optimized/inefficient attack rather than continue to optimize the attack/implementation/etc. If it takes 3 hours to optimize code that will reduce the run-time by 3 seconds, you have passed that point (hint. . . **HINT**).

Project 1-B --- Part 2

- Generate a ***partial*** SHA256 hash collision via brute-force attack



2 Brute-Force Attacks

This portion of the project must be completed individually.

Though brute-force attacks are rarely the best or most-efficient attack, they are always *an attack* that is possible and guaranteed to be successful given sufficient resources. In this portion of the project, you will demonstrate this by implementing a reduced-strength, partial collision attack against a cryptographic hash function (SHA256). You **are not** required to generate a complete collision.

You should be cognizant of the fact that there is often a “point of diminishing returns” where it is more efficient to intentionally perform a non-optimized/inefficient attack rather than continue to optimize the attack/implementation/etc. If it takes 3 hours to optimize code that will reduce the run-time by 3 seconds, you have passed that point (hint. . . **HINT**).

Project 1-B --- Part 2

- Generate a partial SHA256 hash collision via brute-force attack
- Input requirements: prefix w/ AU email
  - `abc1234@auburn.edu`||{anything you want}
- 2 **DIFFERENT** inputs must collide
- Partial collision requirements:
  - Leading 4-bytes are of the digest are identical
  - Both digests leading 4 bytes are identical

Project 1-B --- Part 2

- Generate a partial SHA256 hash collision via brute-force attack
- Input requirements: prefix w/ AU email
  - `abc1234@auburn.edu`||{anything you want}
- 2 **DIFFERENT** inputs must collide
- Partial collision requirements:
  - Leading 4-bytes are of the digest are identical
  - Both digests leading 4 bytes are identical

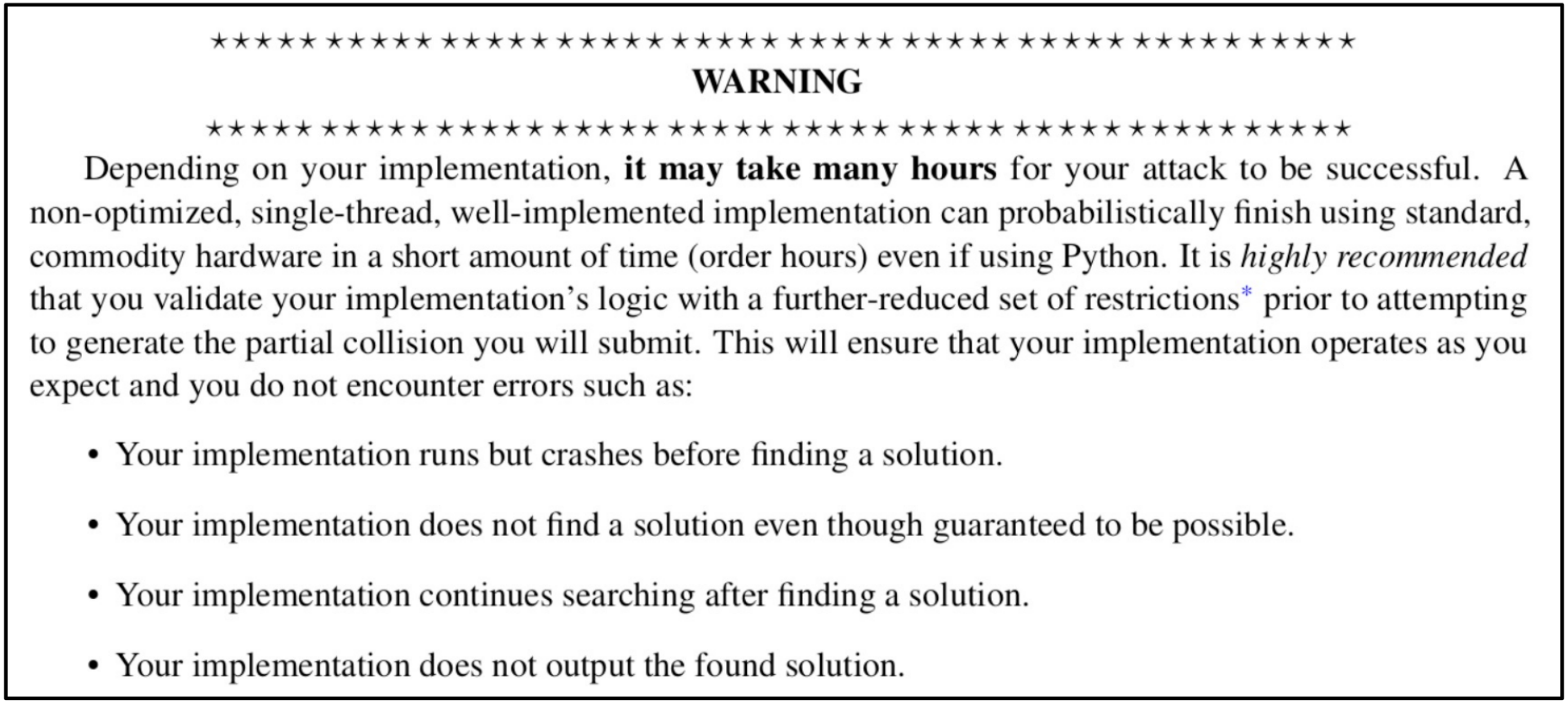
HEX ENCODING IS 2 CHARACTERS PER BYTE

Project 1-B --- Part 2

- **GOOD EXAMPLE:**
  - Digest 1: 0XAAAAAAAA04FDEAB…
  - Digest 2: 0XAAAAAAAA9FDABC3…
- **BAD EXAMPLE 1:**
  - Digest 1: 0XAABBCCDD04FDEAB…
  - Digest 2: 0XAABBCCDD9FDABC3…
- **BAD EXAMPLE 2:**
  - Digest 1: 0XAAAAAAAA04FDEAB…
  - Digest 2: 0XBBBBBBBBFDABC3…

Project 1-B --- Part 2

PLEASE READ THE ASSIGNMENT CAREFULLY TO AVOID ISSUES



★★★★★ ★★★★★ ★★★★★★★★★★ ★★★★★ ★★★★★ ★★★★★ ★★★★★ ★★★★★ ★★★★★★★★★★

WARNING

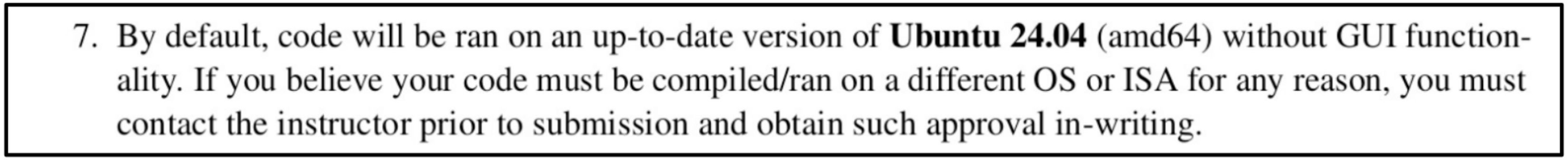
★★★★★ ★★★★★★★★★★ ★★★★★ ★★★★★ ★★★★★ ★★★★★ ★★★★★ ★★★★★★★★★★★★★★★

Depending on your implementation, **it may take many hours** for your attack to be successful. A non-optimized, single-thread, well-implemented implementation can probabilistically finish using standard, commodity hardware in a short amount of time (order hours) even if using Python. It is *highly recommended* that you validate your implementation's logic with a further-reduced set of restrictions[*] prior to attempting to generate the partial collision you will submit. This will ensure that your implementation operates as you expect and you do not encounter errors such as:

- Your implementation runs but crashes before finding a solution.
- Your implementation does not find a solution even though guaranteed to be possible.
- Your implementation continues searching after finding a solution.
- Your implementation does not output the found solution.

Project 1-B --- Part 2

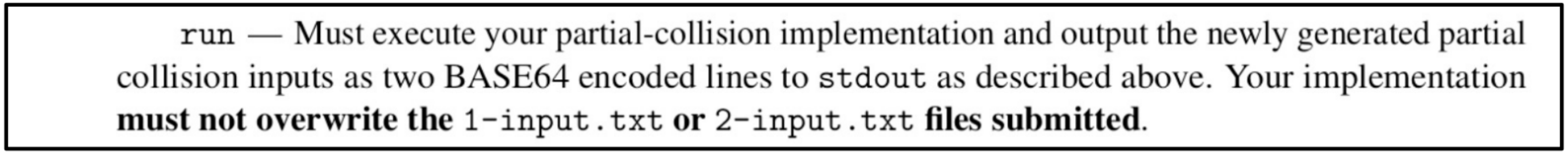
PLEASE READ THE ASSIGNMENT CAREFULLY TO AVOID ISSUES



7. By default, code will be ran on an up-to-date version of **Ubuntu 24.04** (amd64) without GUI functionality. If you believe your code must be compiled/ran on a different OS or ISA for any reason, you must contact the instructor prior to submission and obtain such approval in-writing.

Project 1-B --- Part 2

PLEASE READ THE ASSIGNMENT CAREFULLY TO AVOID ISSUES



`run` — Must execute your partial-collision implementation and output the newly generated partial collision inputs as two BASE64 encoded lines to `stdout` as described above. Your implementation **must not overwrite the** `1-input.txt` **or** `2-input.txt` **files submitted**.

Exam 1 on Tuesday

- During class time, on paper, bring a pen

- Multiple choice, True/False, Matching, etc.
- **Short-answers must be short**
  - Will have an anticipated length to give you idea of *how short* your answers should be

- Bonus available but low point value

Computer and Network Security

Lecture 07:
Sender Authenticity

COMP-5370/6370
Fall 2024