Security Potpourri!

CS 161: Computer Security

Guest Lecturers: Frank Li, Rebecca Portnoff, Grant Ho, Rishabh Poddar

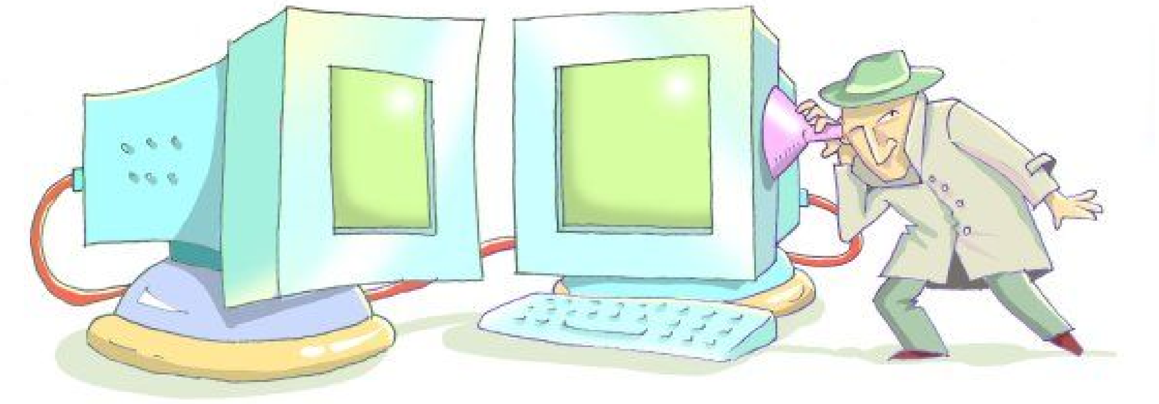
Instructor: Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula, David Fifield, Mia Gil Epner, David Hahn, Warren He, Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic, Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

April 27, 2017

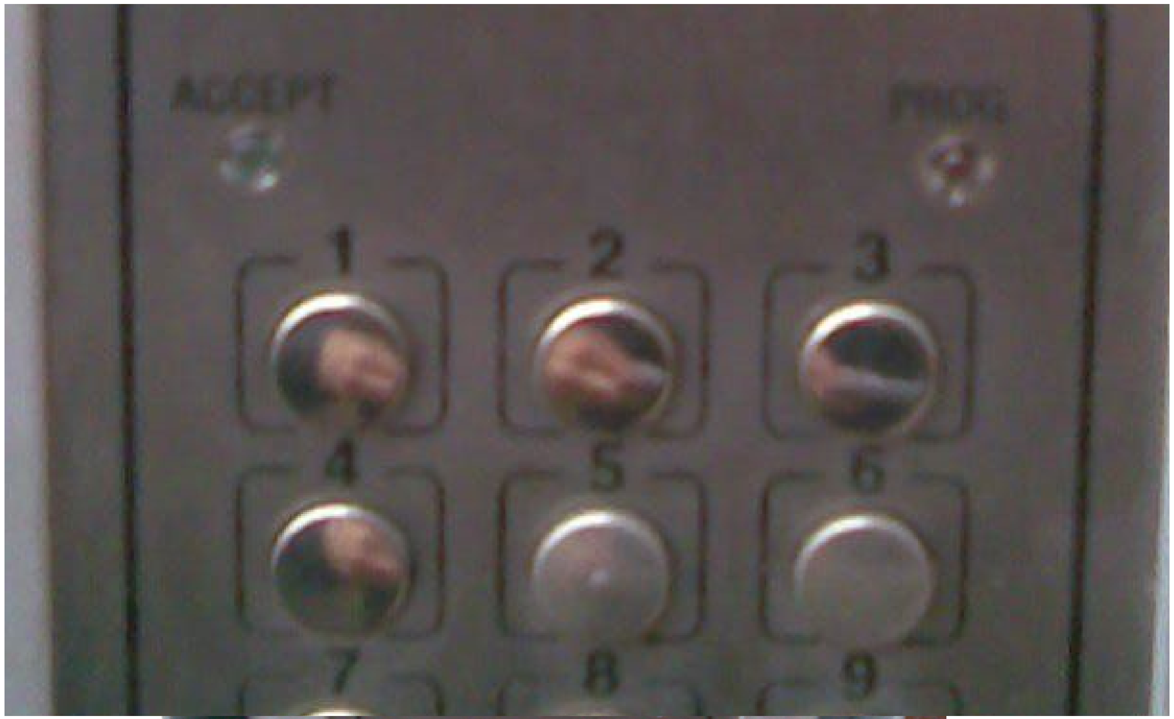
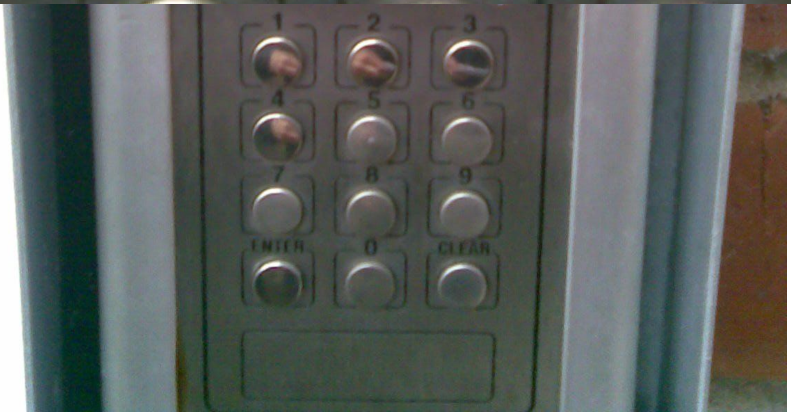
Side Channel Attacks



Side Channels

- Security systems are implemented in software or hardware on physical devices, which interact with their environment.
- Sometimes, attackers can monitor or affect these physical interactions, leaking useful “side channel” information.
- **Side channel attacks** use this information.

Note: Hard to identify due to abstractions.


ACCEPT
PROG
1
2
3
4
5
6
7
8
9

1
2
3
4
5
6
7
8
9
ENTER
0
CLEAR

Attacking Password Checker

```
/* Tenex (old OS) system call to check if submitted password is correct. */

bool CheckPassword(char* submitted_password, char* user){
      char* real_password = GetUserPassword(user);
      for (int i = 0; submitted_password[i] && real_password[i]; ++i) {
            if (submitted_password[i] != real_password[i])
                  return False;
      }
      /* Ensures both strings are same len. */
      return submitted_password[i] == real_password[i];
}
```

Attacking Password Checker

```
/* Tenex (old OS) system call to check if submitted password is correct. */

bool CheckPassword(char* submitted_password, char* user){
    char* real_password = GetUserPassword(user);
    for (int i = 0; submitted_password[i] && real_password[i]; ++i) {
        if (submitted_password[i] != real_password[i])
            return False;
    }
    /* Ensures both strings are same len. */
    return submitted_password[i] == real_password[i];
}
```

Attacking Password Checker

```
/* Tenex (old OS) system call to check if submitted password is correct. */

bool CheckPassword(char* submitted_password, char* user){
    char* real_password = GetUserPassword(user);
    for (int i = 0; submitted_password[i] && real_password[i]; ++i) {
        if (submitted_password[i] != real_password[i])
            return False;
    }
    /* Ensures both strings are same len. */
    return submitted_password[i] == real_password[i];
}
```

Attacking Password Checker

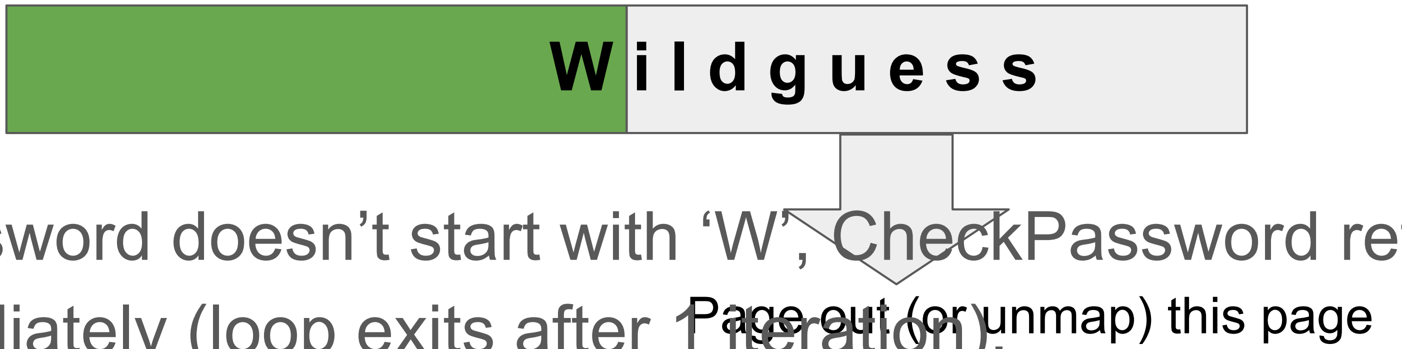
```
/* Tenex (old                                              correct. */

bool CheckP
    char* re
    for (int i                                             ) {
        if (su

    }
    /* Ensur
    return submitted_password[i] == real_password[i];
}
```

Say passwords are only alphanumeric. To brute force a 10-character password, requires guessing:

$62^{10} = 8.39*10^{17}$ possible passwords.

Better “Side Channel” Attack

Leverage memory layout of the submitted password, by spreading it out across multiple pages.



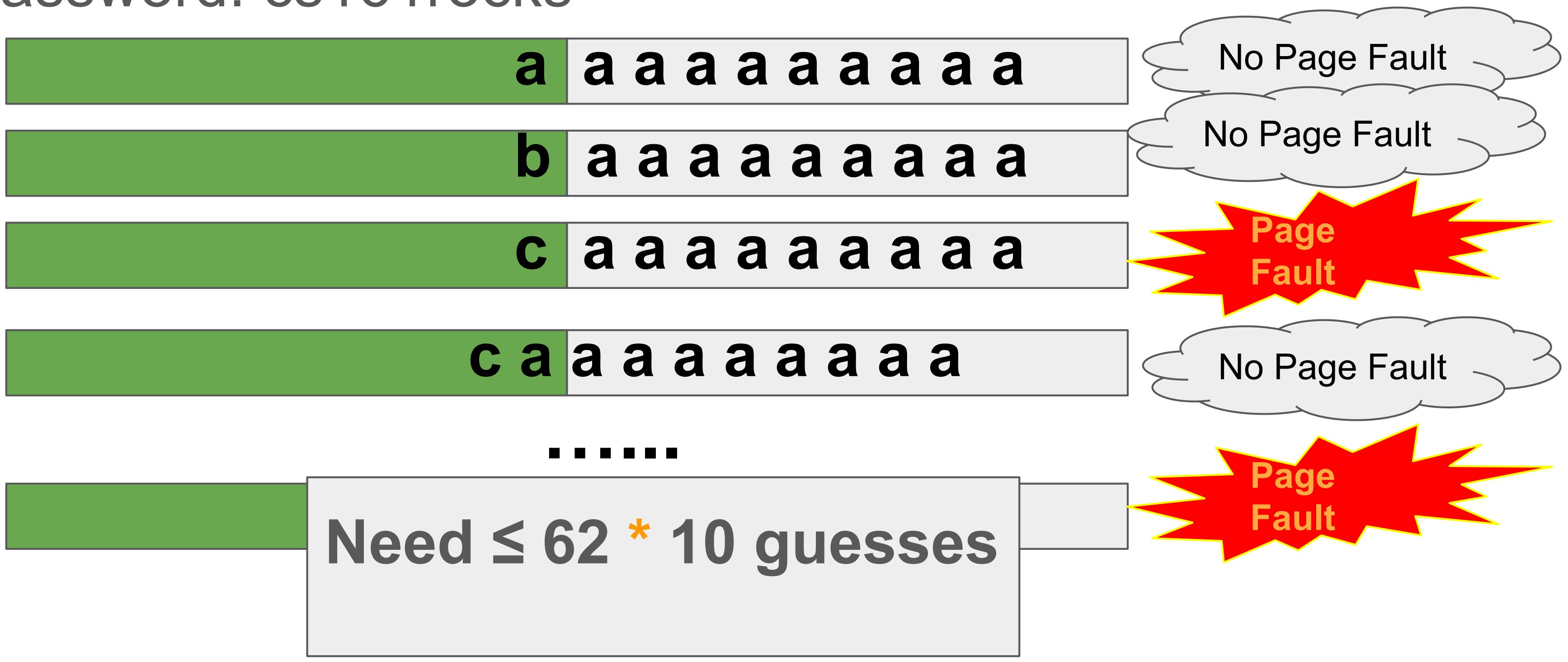
If password doesn’t start with ‘W’, CheckPassword returns immediately (loop exits after 1 iteration).

If password **DOES** start with ‘W’, CheckPassword looks for second character of submitted password, and **page faults!**

Better “Side Channel” Attack

Page faults are slow, *timing side channel!* (Seg faults also visible)

Real password: cs161rocks



Potential Fixes?

Fix 1: Always check entire password.

- Might still leak password length based on how long the check takes!

Fix 2: Assume a max length for password. Always loop that many times, even if password is shorter.

- Constant time algorithm: Eliminates timing side channel, but now caps password length and has worse performance.

Power Analysis on RSA

RSA decryption: $M = C^d \bmod N$

Common algorithm for exponentiation is “square and multiply”.

```
def exponentiate(base C, exponent d):
    V = 1
    For each bit b in d (most to least significant):
        V = V^2 mod N
        If b==1: V = V*C mod N
    return V
```

Ex: d=1010 in binary = 10 in decimal.

Old V = 1

Power Analysis on RSA

RSA decryption: $M = C^d \bmod N$
Common algorithm for exponentiation is “square and multiply”.

```
def exponentiate(base C, exponent d):
    V = 1
    For each bit b in d (most to least significant):
        V = V^2 mod N
        If b==1: V = V*C mod N
    return V
```

Ex: d=**1**010 in binary = 10 in decimal.
Old V = 1, New V = $1^2 * C = C$ (bit is 1)

Power Analysis on RSA

RSA decryption: $M = C^d \bmod N$

Common algorithm for exponentiation is “square and multiply”.

```
def exponentiate(base C, exponent d):
    V = 1
    For each bit b in d (most to least significant):
        V = V^2 mod N
        If b==1: V = V*C mod N
    return V
```

Ex: d=1**0**10 in binary = 10 in decimal.

Old V = C, New V = C^2 (bit is 0)

Power Analysis on RSA

RSA decryption: $M = C^d \bmod N$

Common algorithm for exponentiation is “square and multiply”.

```
def exponentiate(base C, exponent d):
    V = 1
    For each bit b in d (most to least significant):
        V = V^2 mod N
        If b==1: V = V*C mod N
    return V
```

Ex: d=10**1**0 in binary = 10 in decimal.

Old $V = C^2$, New $V = C^{2*2} * C = C^5$ (bit is 1)

Power Analysis on RSA

RSA decryption: $M = C^d \bmod N$

Common algorithm for exponentiation is “square and multiply”.

```
def exponentiate(base C, exponent d):
    V = 1
    For each bit b in d (most to least significant):
        V = V^2 mod N
        If b==1: V = V*C mod N
    return V
```

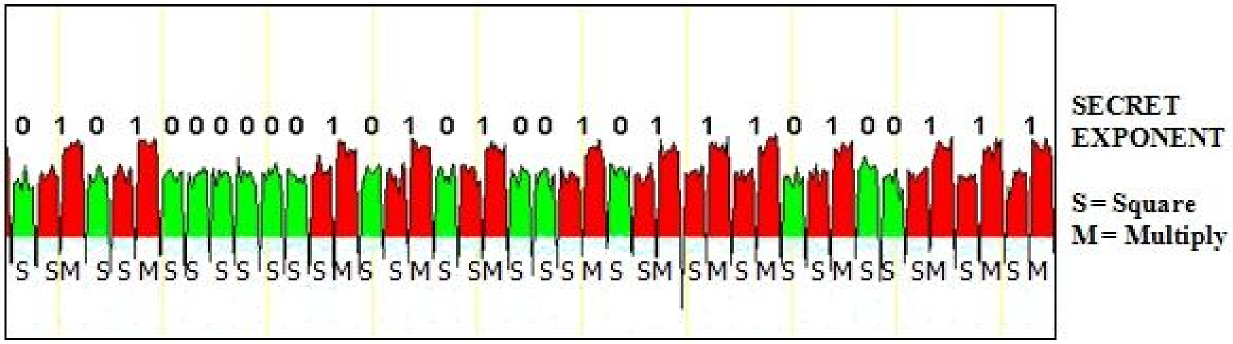
Ex: d=101**0** in binary = 10 in decimal.

Old $V = C^5$, New $V = C^{10}$ (bit is 0), **as expected!**

Power Analysis on RSA

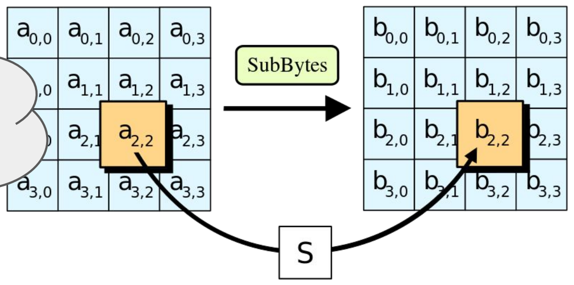
RSA decryption: M = C^d mod N
Common algorithm for exponentiation is "square and multiply".

Square + multiply computation produces different power usage profile than just squaring! Can distinguish between a 0 or 1 bit in secret key based on power usage!



AES Cache Timing Attack

Did not cover in lecture, just for reference!


SubBytes
S
$a_{0,0}$ $a_{0,1}$ $a_{0,2}$ $a_{0,3}$
$a_{1,1}$ $a_{1,2}$ $a_{1,3}$
$a_{2,1}$ $a_{2,2}$ $a_{2,3}$
$a_{3,0}$ $a_{3,1}$ $a_{3,2}$ $a_{3,3}$
$b_{0,0}$ $b_{0,1}$ $b_{0,2}$ $b_{0,3}$
$b_{1,0}$ $b_{1,1}$ $b_{1,2}$ $b_{1,3}$
$b_{2,0}$ $b_{2,1}$ $b_{2,2}$ $b_{2,3}$
$b_{3,0}$ $b_{3,1}$ $b_{3,2}$ $b_{3,3}$

AES's computation accesses tables of values. Which indices are accessed is based on the secret key.

If these tables are stored in memory shared by the attacker and victim process (e.g. memory deduplication), attacker can load the tables into cache, *except* for one index. Later attacker can load that missing index.

- Cache miss = slower, victim didn't use the missing index
- Cache hit = faster, victim used the missing index

Can learn the secret key based on indices accessed.

Other side channels used for attacks

- Timing
- Cache hits
- Power usage
- Electromagnetic radiation
- Acoustics
- Optical
- Data remanence (“deleted” but uncleared memory)
- Row Hammer (change off-limit memory by accessing adjacent memory)
- Network side channels (recall global IP ID scanning from Homework 4)

Backpage and Bitcoin: Uncovering Human Traffickers

Bitcoin

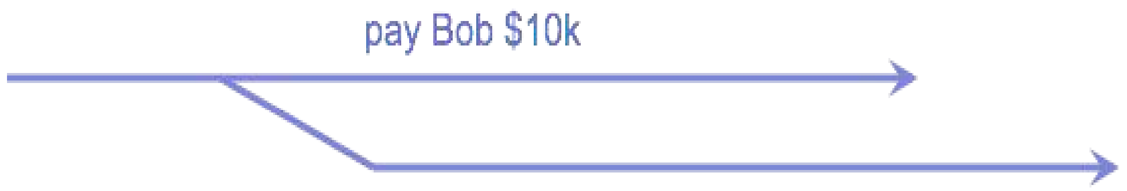
- Bitcoin ownership is pseudonymous
  - Exchange bitcoin using pseudonyms
  - Pseudonyms are public keys, tied to private key the user owns
  - Sign out-going transactions with private key

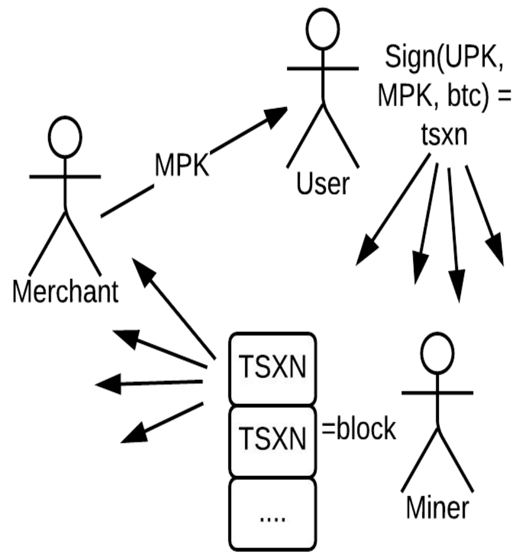
Blockchain

- Public, distributed, peer-to-peer, hash-chained audit log of all transactions
    - Hash chain is public, broadcasted on peer-to-peer network, and append-only

Blockchain cont'd

- How do you get bitcoin?
  - Mining
    - Append block to most recent/longest version of blockchain
  - Buy it





Sex Trafficking and the Internet

- Internet has opened new ways for traffickers to advertise and find victims
- Broader goal : use computer science tools/techniques to fight sex trafficking and slavery
- Detect traffickers from advertisements they pay for and post

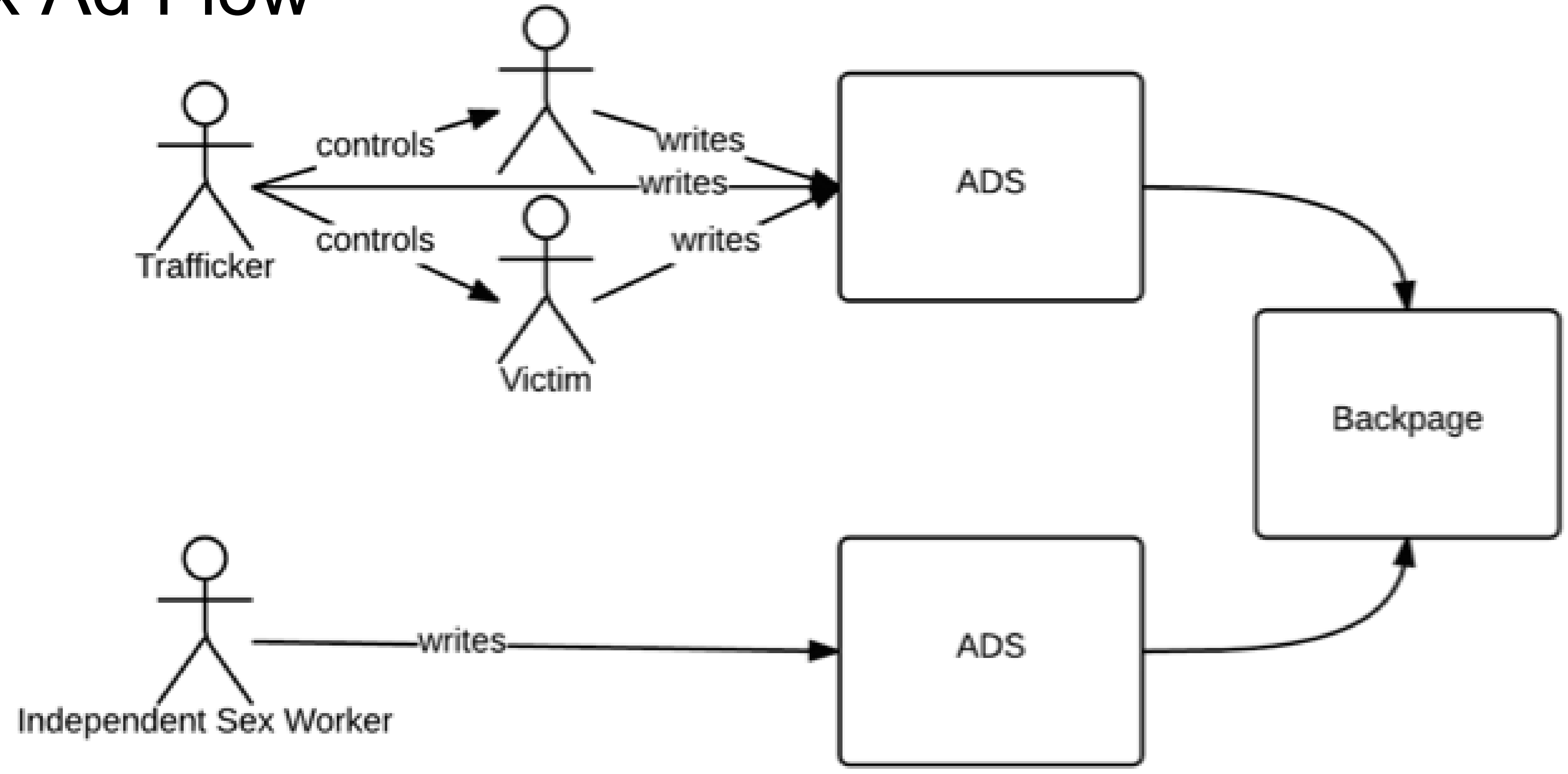
Problem Statement

- Can I distinguish traffickers from independent sex workers on classified ad sites?
  - Too much data

Backpage

- 2nd largest online classified ad site in the US
- 80% percent of the market for online sex ads in USA
- Running since 2004, listings all over the world
- Used by traffickers to advertise their victims
- Two forms of payment for adult entertainment listings
  - Bitcoin
  - Check/money order sent via regular mail

Sex Ad Flow



controls
writes
writes
controls
writes
Trafficker
Victim
ADS
Backpage
writes
Independent Sex Worker
ADS

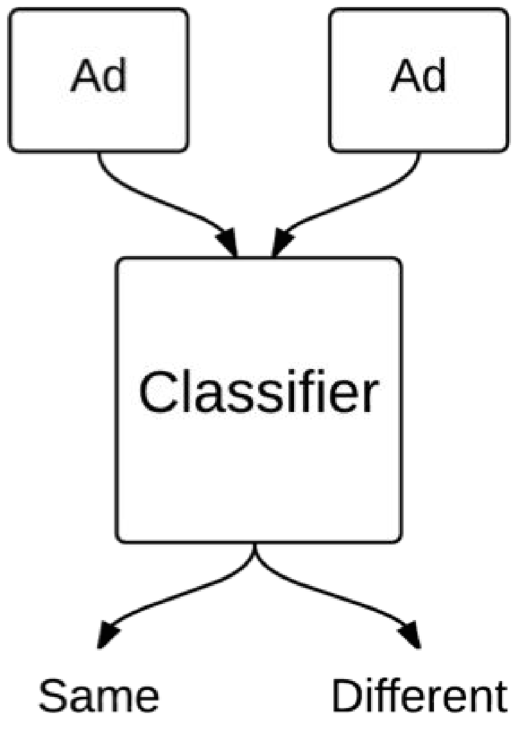
Goal

- Develop techniques to cluster sex ads by owner
  - Current best clustering is via hard-link; unreliable

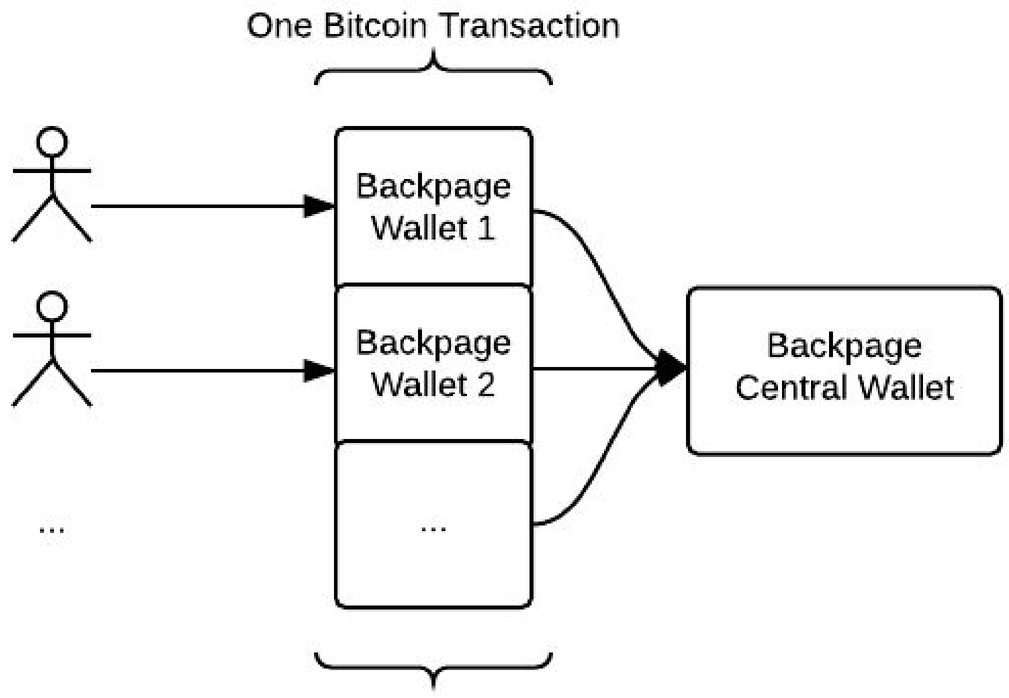
Results

- Two different methodologies that combine the classifier, linking technique and existing hard identifiers to group ads by owner
  - Stylometry classifier that distinguishes between sex ads posted by the same vs. different authors with 90% TPR and 1% FPR
  - Side channel attack that takes advantage of leakages from the Bitcoin blockchain and sex ad site to link a subset of sex ads to Bitcoin public wallets and transactions
- Analyzed 4-weeks of scraped sex ads from Backpage
  - Rebuild the price of each Backpage sex ad, and analyze the output of the two different methodologies

Stylometry model

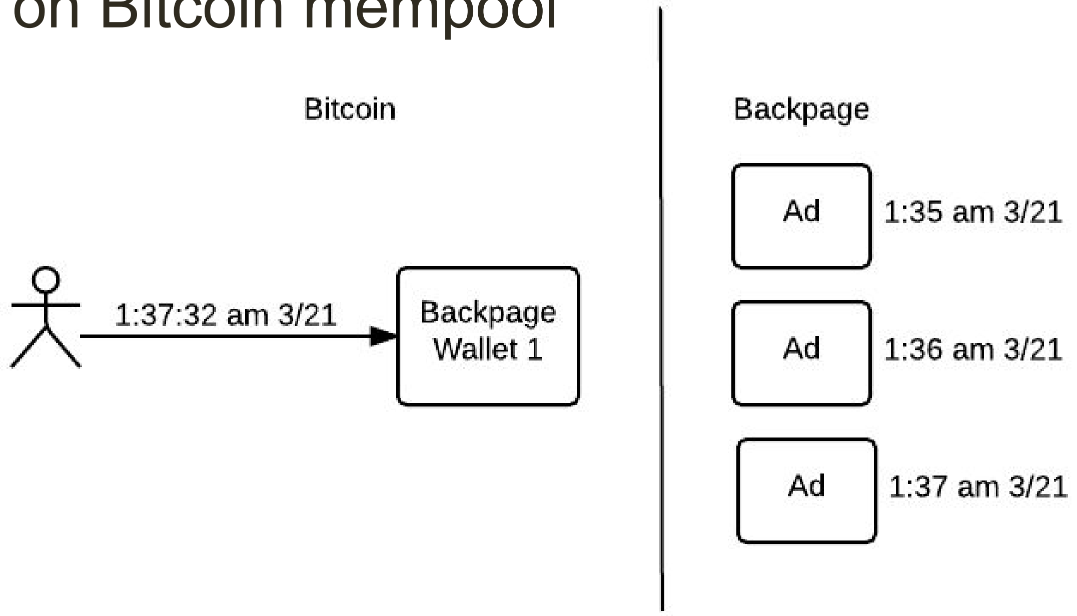


Backpage Payment Flow



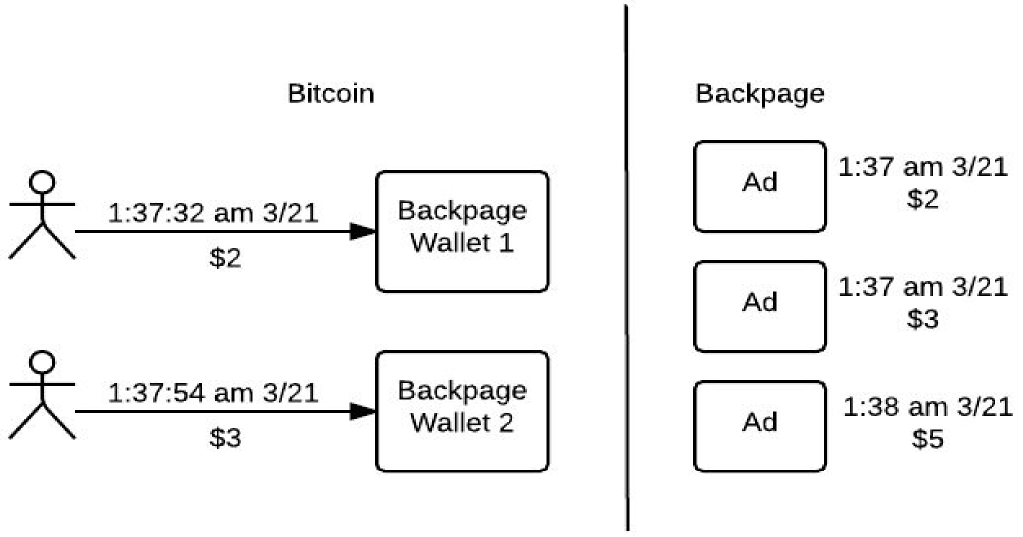
Timing Side Channel Attack

- Backpage posts ad onto its site one minute after payment appears on Bitcoin mempool



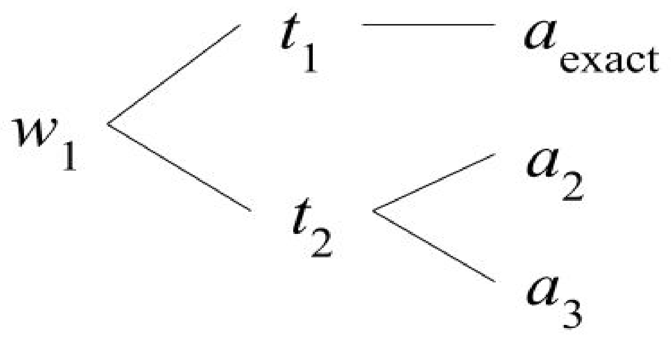
Timing & Price Side Channel Attack

- Backpage's pricing algorithm takes ad posting frequency and location as variables, and can be reverse engineered



Persistent Bitcoin Identity Methodology

- Goal: map each ad to its true owner wallet
- Persistent Bitcoin Identity: any wallet that sends the change from each of its transactions back into itself, and has one exact match
    - Use stylometry model to distinguish non-exact match
    - All ads that match to this wallet are clustered under this PBI



$$\begin{tikzcd}
 & t_1 \arrow[r, dash] & a_{\text{exact}} \\
w_1 \arrow[ur, dash] \arrow[dr, dash] & & a_2 \\
 & t_2 \arrow[ur, dash] \arrow[dr, dash] & \\
 & & a_3
\end{tikzcd}$$

4-week case study

- 26 ‘ground truth’ test ads
  - 25 required payment, 1 free
  - Placed from Dec 12th, 2016 to Dec 24th, 2016
  - Price range from $2 to $20
  - Posted in 27 distinct US regions
- Scraped all the sex ads in every US location every hour, for 4 weeks
- 741,443 unique ads scraped
  - 151,482 required payment
  - Placed from Dec 10th, 2016 to Jan 9th, 2017
  - Price range from $1 to >$100
  - Posted in 60 distinct US regions

4-week case study: PBI

- 11 ground truth ads paid using a PBI
  - 8 transactions were exact match for correct ad
  - 3 transactions matched two ads, one of which was the correct ad
- 249 PBI’s total
- 90 of those PBI’s had at least one exact match
- Results:
  - Links between hard identifiers
  - Evidence of networks across multiple locations
  - Owners of sex ad clusters spending a lot of money on ads

Conclusion

- Promising!
- First work to try to link specific purchases to specific transactions on the Blockchain
- Lots of work left to be done

User Authentication & Passwords

CS 161: Computer Security

Guest Lecturer: Grant Ho

Instructor: Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula, David Fifield, Mia Gil Epner, David Hahn, Warren He, Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic, Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

April 27, 2017

With content from Raluca Ada Popa & Dan Boneh

Attacks & Defenses on Password Authentication

- Often worry about 3 classes of attacks (threat models)
  1. Online guessing
  2. Server compromise (“offline guessing”)
  3. Client password compromise

Attacks & Defenses on Password Authentication

- Often worry about 3 classes of attacks (threat models)
  1. Online guessing
  2. Server compromise ("offline guessing")
  3. Client password compromise

- We'll just focus on the last two threat models b/c of time constraints

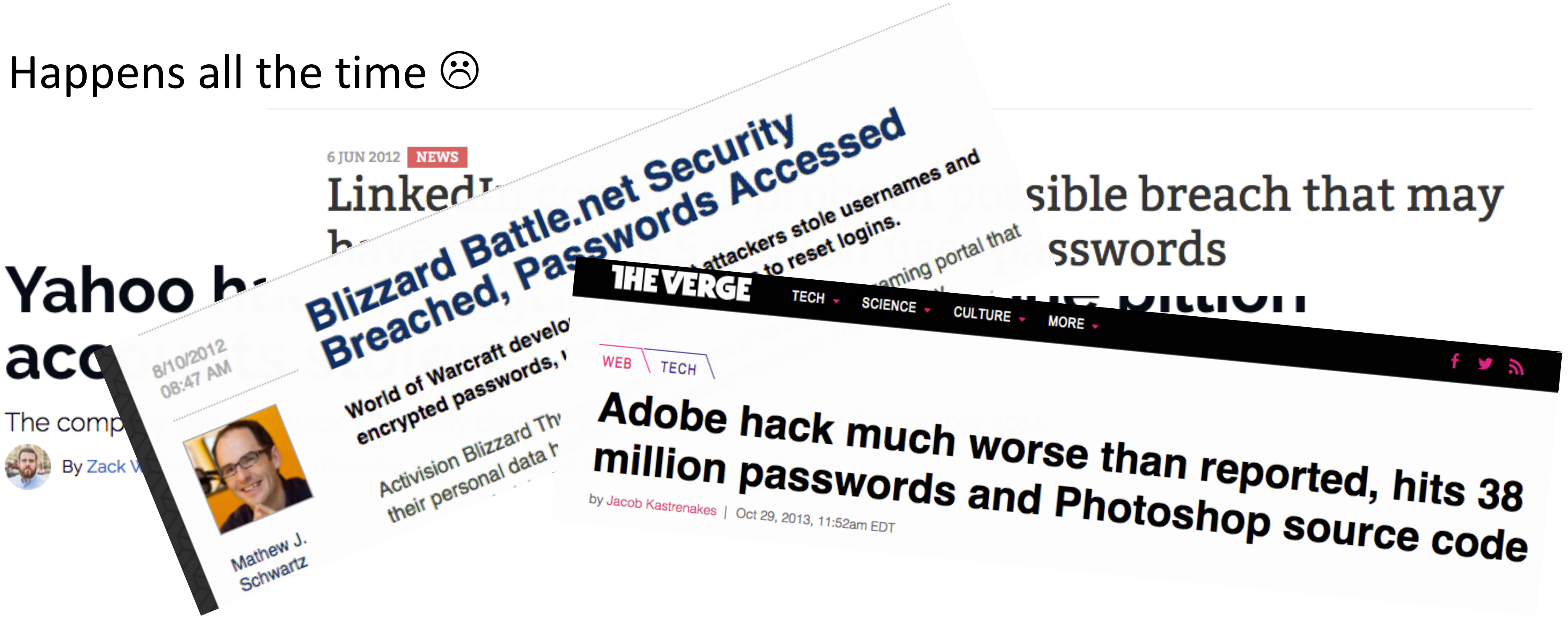
Threat Model 1:
Server Compromise

Attacker breaks into server and steals password database

(also called “offline guessing attacks”)

Threat Model #1: Server Compromised

- Attacker breaks into server and steals password database
- Happens all the time ☹



6 JUN 2012 NEWS

LinkedI… …sible breach that may … …sswords

Yahoo h… acc… …e billion

The comp…

By Zack W…

8/10/2012 08:47 AM

Mathew J. Schwartz

Blizzard Battle.net Security Breached, Passwords Accessed

World of Warcraft develo… encrypted passwords, …

…attackers stole usernames and … to reset logins.

Activision Blizzard Th… their personal data h…

…ming portal that

THE VERGE TECH SCIENCE CULTURE MORE

WEB TECH

Adobe hack much worse than reported, hits 38 million passwords and Photoshop source code

by Jacob Kastrenakes | Oct 29, 2013, 11:52am EDT

Threat Model #1: Server Compromised

- Insecure Defense: Server stores encrypted passwords in its database
- But server needs easy access to secret key in order to verify users when they login
  - So, if Mallory breaks into the server, then she can just steal secret key too!

Encrypting passwords is ***not*** a secure solution

Secure Password Storage

- Server should store ***salted + hashed*** passwords (Section 6, Problem #1)
  - **Setup**
    1. During account registration, server generates random number (salt)
    2. Server computes ***h*** = hash(salt, password)
    3. Server stores (username, salt, ***h***) and *deletes user's password*

username	salt	***h*** = hash(salt, password)
Alice	235545235	Hash(Alice's pwd, 235545235)
Bob	678632523	Hash(Alice's pwd, 678632523)

  - **Authentication**
    - User's browser sends {username, password} to server
    - Server computes hash(salt, password) and checks if it matches ***h***

Secure Password Storage

- **Secure Defense**: Server should store ***salted + hashed*** passwords

username	salt	h = hash(salt, password)
Alice	235545235	Hash(Alice’s pwd, 235545235)
Bob	678632523	Hash(Alice’s pwd, 678632523)

- Attacker steals password database, but:
  - Only sees salts & *h’s*: salt is random & secure hash functions are one-way.
  - Attacker can still compute big table of guesses for Alice & check for matching ***h***:

‘123456’	**Hash(‘123456’, 235545235)**
‘password’	Hash(‘password’, 235545235)
‘aaaaaaaa’	Hash(‘aaaaaaa’, 235545235)
…	…

But salting forces attacker to re-compute table for each user and prevents pre-computation.

Secure Password Storage

- **Secure Defense**: Server should store *salted* **+** *securely hash* passwords

- The secure hash function should also be ***slow*** to compute
  - Usually we want fast crypto for performance
  - But here we want attacker to wait… and wait… and wait… for guessing to succeed.
  - Examples: Argon, bcrypt, scrypt

- Conceptually, Slow-Hash(x) = hash(hash(hash(hash(…(hash(x)))))
  - where hash is a regular secure hash (e.g., SHA-256 or HMAC)

- If Slow-Hash is 1,000 times slower, attack that previously took 1 day now takes ~3 years

Threat Model 2:
Client Password Compromise

Attacker obtains Alice’s password

- Phishing
- Surveillance camera (airport, cafe, etc.) records Alice typing password
- …

Threat Model #2: Password Compromise

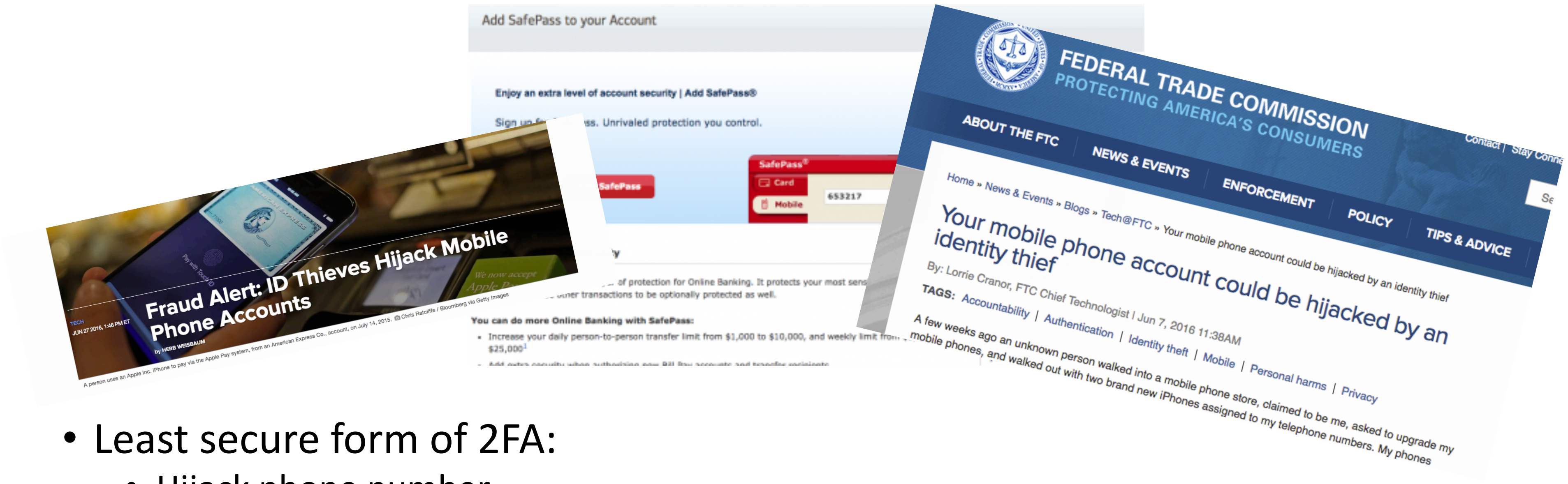
- Defense: Two-factor authentication (2FA)
  1. Something you knows (password)
  2. Something you have (smartphone/authentication device)
  3. Something you are (fingerprints/iris scanner)
  - Require 2 methods from above

- Most common 2FA: password + authentication device
  - User enters password at login
  - If password correct, user then needs to use authentication device

- Let’s examine some 2FA designs for the authentication device

Common 2FA Designs

1. Text message: server generates random number & texts it to you



Fraud Alert: ID Thieves Hijack Mobile Phone Accounts

TECH JUN 27 2016, 1:46 PM ET by HERB WEISBAUM

A person uses an Apple Inc. iPhone to pay via the Apple Pay system, from an American Express Co. account, on July 14, 2015. Chris Ratcliffe / Bloomberg via Getty Images

Add SafePass to your Account

Enjoy an extra level of account security | Add SafePass®

Sign up f... ss. Unrivaled protection you control.

SafePass® Card Mobile 653217

...of protection for Online Banking. It protects your most sens... ...other transactions to be optionally protected as well.

You can do more Online Banking with SafePass:

- Increase your daily person-to-person transfer limit from $1,000 to $10,000, and weekly limit from ... $25,000[1]

FEDERAL TRADE COMMISSION PROTECTING AMERICA'S CONSUMERS

ABOUT THE FTC | NEWS & EVENTS | ENFORCEMENT | POLICY | TIPS & ADVICE

Home » News & Events » Blogs » Tech@FTC » Your mobile phone account could be hijacked by an identity thief

Your mobile phone account could be hijacked by an identity thief

By: Lorrie Cranor, FTC Chief Technologist | Jun 7, 2016 11:38AM

TAGS: Accountability | Authentication | Identity theft | Mobile | Personal harms | Privacy

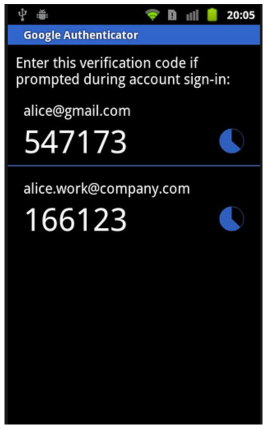
A few weeks ago an unknown person walked into a mobile phone store, claimed to be me, asked to upgrade my mobile phones, and walked out with two brand new iPhones assigned to my telephone numbers. My phones

- Least secure form of 2FA:
  - Hijack phone number
  - Mobile malware or any app w/ text message permissions (e.g. Tinder, Uber, etc.)
  - …

Common 2FA Designs

1. **Text message: server generates random number & texts it to you**
    - Least secure: hijack phone number or hack telephone company (e.g., nation state)

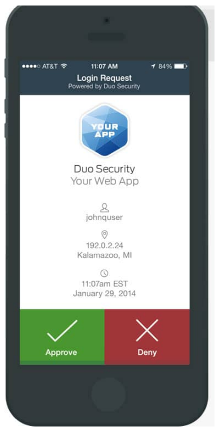
2. **Authenticator apps (more secure)**
    - Google Authenticator, Duo, etc.
    - Protocols:
        - S/KEY
        - TOTP
        - Push notification



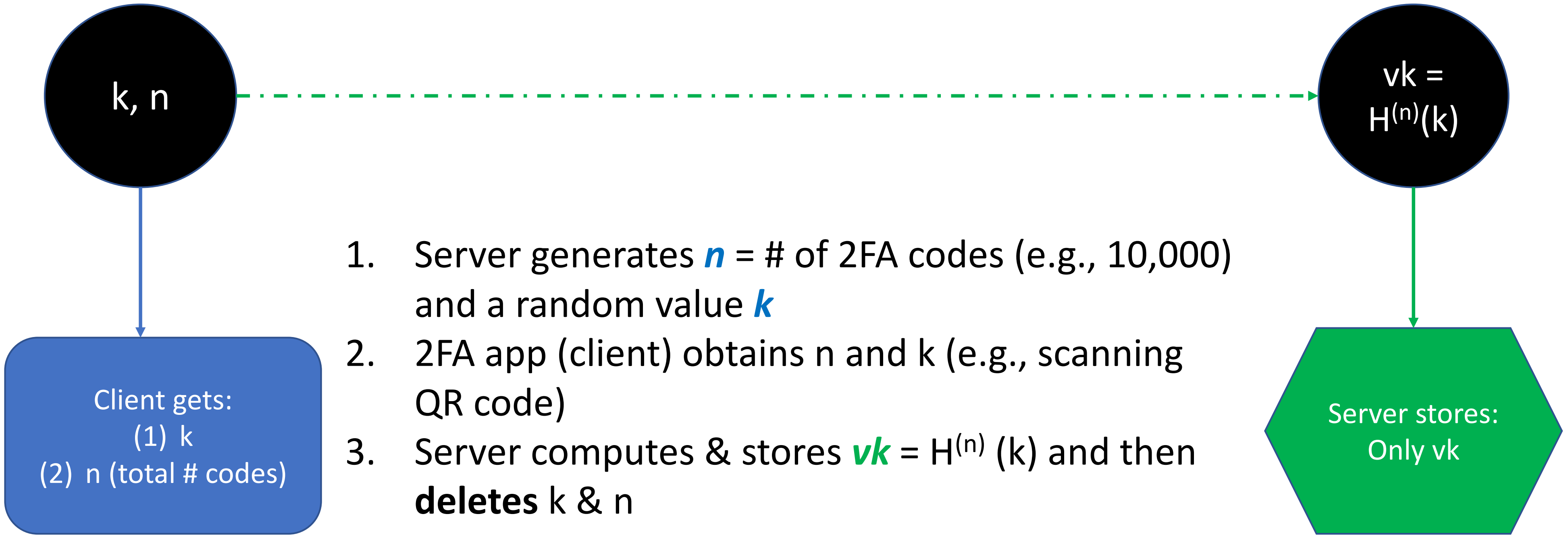
Enter this verification code if prompted during account sign-in:

alice@gmail.com
547173

alice.work@company.com
166123



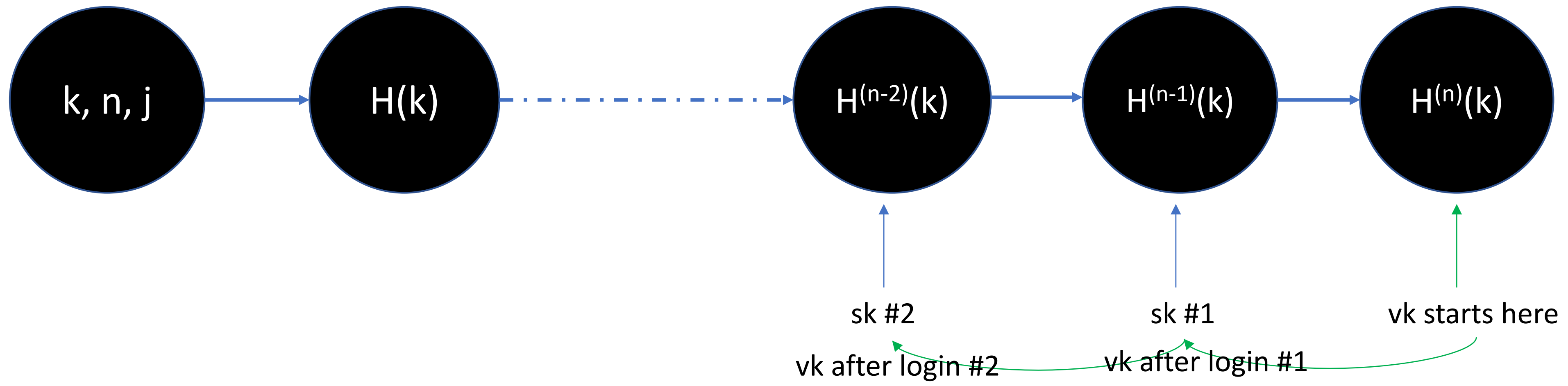
The S/Key Protocol: Setup



1. Server generates ***n*** = # of 2FA codes (e.g., 10,000) and a random value ***k***
2. 2FA app (client) obtains n and k (e.g., scanning QR code)
3. Server computes & stores ***vk*** = $H^{(n)}$ (k) and then **deletes** k & n

$H^{(n)}$ (k) = H(H(H(…(H(k)))), hash for n times

The S/Key Protocol: Authenticating to Server



Client stores:
(1) k
(2) n (total # codes)
(3) j (# total logins)

1. Client computes & sends sk = $H^{(n-j)}(k)$
2. Server checks if H(sk) = vk
3. Server updates vk = sk
4. Repeat 1-3

Secure even if attacker breaks into server and steals vk for each user!

Server stores:
Only vk

Common 2FA Designs

1. **Text message: server generates random number & texts it to you**
   - Least secure: hijack phone number or hack telephone company (e.g., nation state)

2. **Authenticator apps (more secure)**
   - Common protocols: S/KEY, TOTP, Push notification
   - Still vulnerable to phishing!
     1. Phishing page asks for user's password
     2. Next, phishing page asks user to enter 2FA code
     3. Attacker then uses both to login

Common 2FA Designs

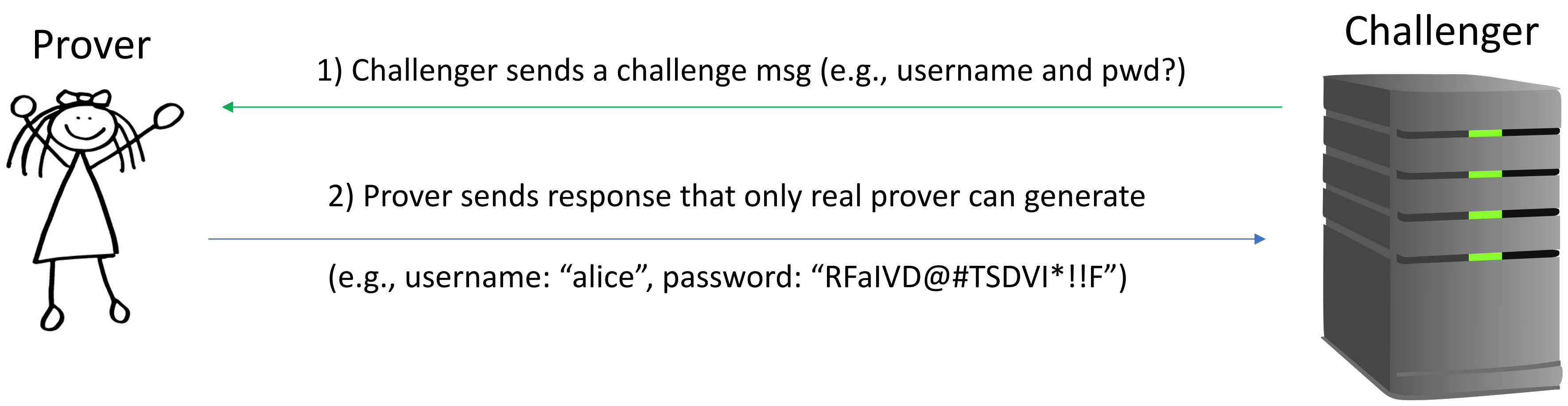
1. **Text message: server generates random number & texts it to you**
   - Least secure: hijack phone number or hack telephone company (e.g., nation state)

2. **Authenticator app (more secure)**
   - Push notification, S/KEY, TOTP
   - Still vulnerable to phishing!

3. **Hardware tokens: challenge-response (most secure)**
   - Hardware device that's <u>plugged into laptop</u>
   - Can protect against phishing attacks

Challenge-Response (General)

- General protocols for authentication
- A "***prover***" wants to authenticate to a "***challenger***"
  - E.g., a user (prover) wants to login to Gmail (challenger) as Alice



1) Challenger sends a challenge msg (e.g., username and pwd?)

2) Prover sends response that only real prover can generate

(e.g., username: "alice", password: "RFaIVD@#TSDVI*!!F")

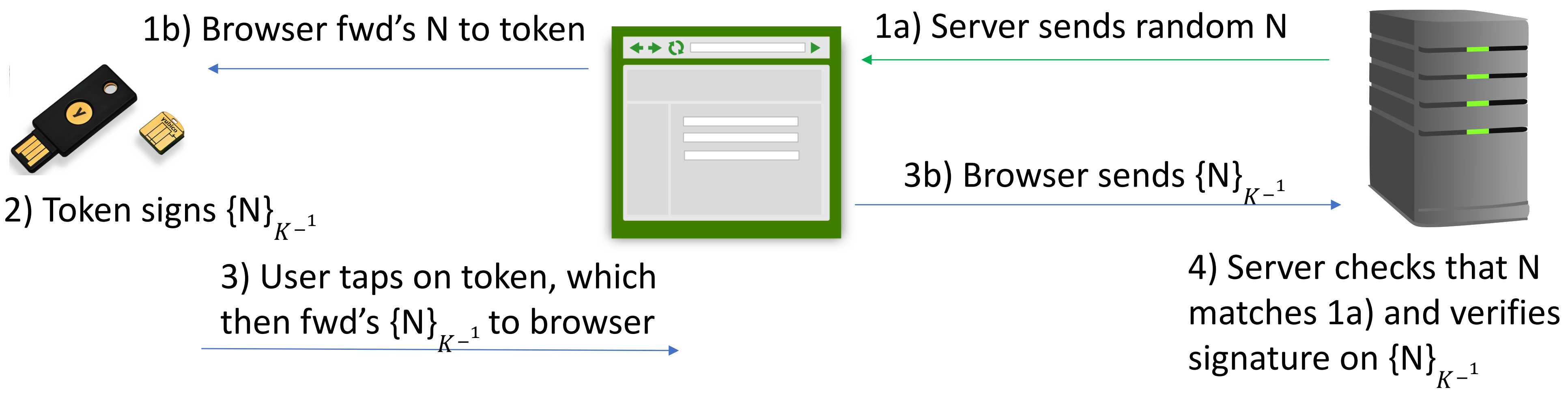
2FA Challenge-Response

- Hardware 2FA token has a public & private key pair embedded in device

A. <u>Setup</u>

1. Alice's browser gets ***K*** = 2FA token's public key and sends *K* to server
2. Server stores (username, *K*) in its 2FA database

B. <u>Authentication</u>

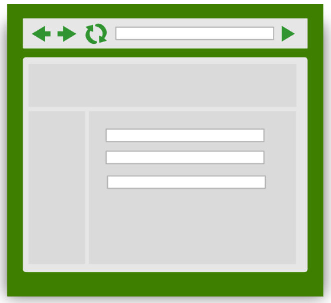

1b) Browser fwd's N to token
1a) Server sends random N
3b) Browser sends $\{N\}_{K^{-1}}$
2) Token signs $\{N\}_{K^{-1}}$
3) User taps on token, which then fwd's $\{N\}_{K^{-1}}$ to browser
4) Server checks that N matches 1a) and verifies signature on $\{N\}_{K^{-1}}$

Adding Phishing Resistance

1b) Browser fwd’s N to token

AND it includes D = domain of actual webpage in browser

1a) Server sends random number N



3b) Browser sends $\{N, D\}_{K^{-1}}$

2) Token signs $\{N, D\}_{K^{-1}}$

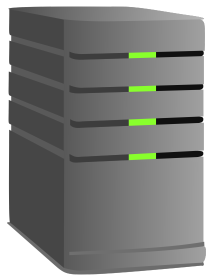
3) User taps on token, which then fwd’s $\{N, D\}_{K^{-1}}$ to browser

4) Server checks:

- D matches its domain
- N matches what it sent
- Valid signature on $\{N, D\}_{K^{-1}}$

Phishing Attack Now Fails!

- During phishing attack, browser will be at website w/ domain D’ = gmai1.com, instead of real domain **D = gmail.com**



Phishing Attack Now Fails!

- During phishing attack, browser will be at website w/ domain D’ = gmai1.com, instead of real domain **D = gmail.com**

1b) Browser fwd’s N to token

AND it includes D’ = domain of actual webpage in browser

1a) Gmail sends random number N

3b) Browser sends $\{N, \textbf{D'}\}_{K^{-1}}$

2) Token signs $\{N, \textbf{D'}\}_{K^{-1}}$

3) User taps on token, which then fwd’s $\{N, \textbf{D'}\}_{K^{-1}}$ to browser

4) Gmail checks:

- N matches what it sent
- Valid signature on $\{N, \textbf{D'}\}_{K^{-1}}$
- **But D’** doesn’t match its domain!

Practical Advice for Future Security Engineers

Applicable to your users and your employees:

1. Use HTTPS (prevent MITM from seeing passwords)
2. Securely store passwords (Threat Model #1)
3. Enable 2FA, ideally hardware tokens (Threat Model #2)
4. Securely check passwords & rate limit (not covered b/c of time)
5. Incorporate detection systems if you can (not covered b/c of time)
    1. Access logging
    2. Spearphishing detection
    3. Honey accounts/Tripwire

Computing on private data

CS 161: Computer Security

Instructor: Prof. Vern Paxson

TAs: Paul Bramsen, Apoorva Dornadula, David Fifield, Mia Gil Epner, David Hahn, Warren He, Grant Ho, Frank Li, Nathan Malkin, Mitar Milutinovic, Rishabh Poddar, Rebecca Portnoff, Nate Wang

http://inst.eecs.berkeley.edu/~cs161/

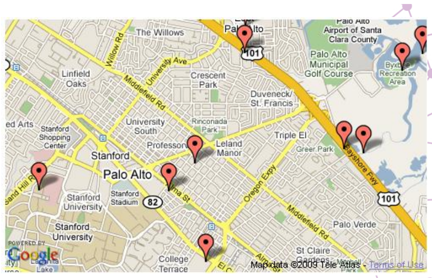
April 27, 2017

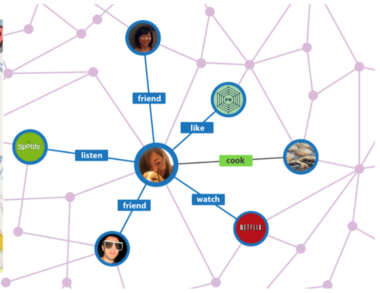
Guest Lecturer: Rishabh Poddar

With content from
Raluca Ada Popa, Dan Boneh, and Taesoo Kim

Many decisions are made on private data

- User data (e.g. email, social)
- Medical data
- Financial data
- Location data

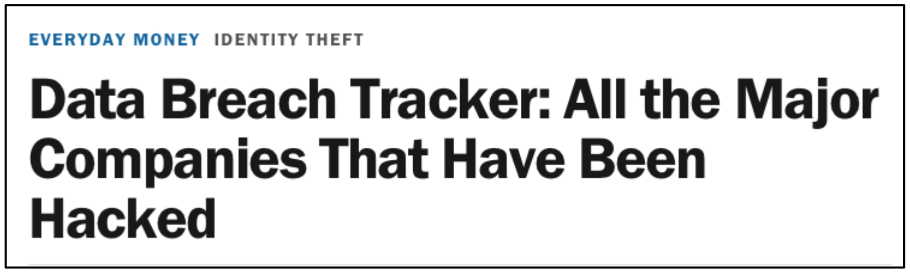




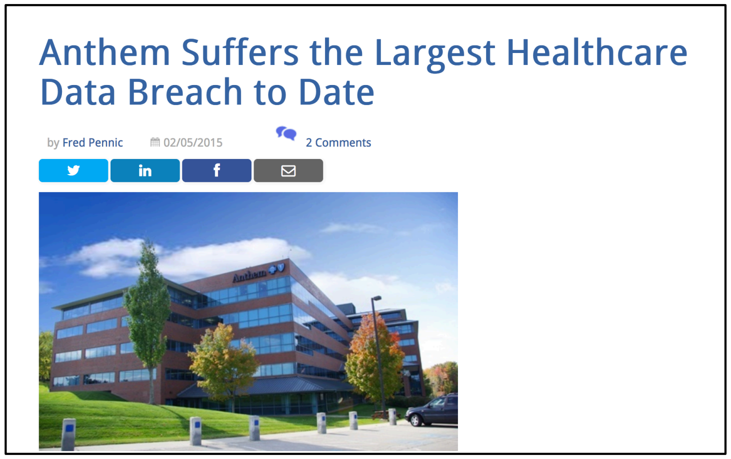
Data stored unencrypted in order to allow applications to compute queries / make decisions

Defense: try to build ***walls*** around the data (e.g. access control, firewalls, IDS, etc.)

Attackers eventually break into systems

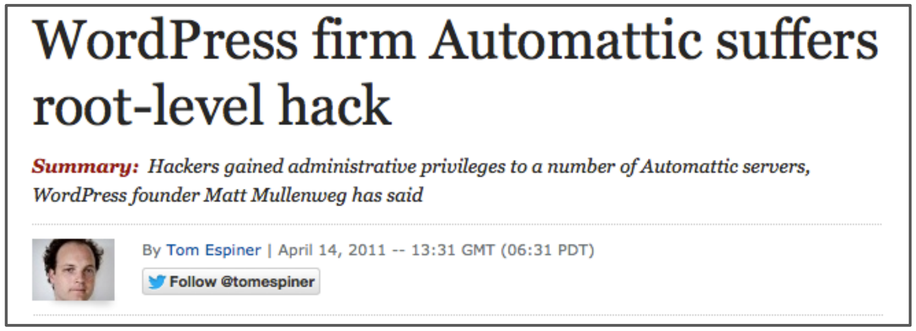

EVERYDAY MONEY IDENTITY THEFT

Data Breach Tracker: All the Major Companies That Have Been Hacked


Anthem Suffers the Largest Healthcare Data Breach to Date

by Fred Pennic 02/05/2015 2 Comments

- Sometimes, they even obtain root access or have admin privilege


WordPress firm Automattic suffers root-level hack

Summary: Hackers gained administrative privileges to a number of Automattic servers, WordPress founder Matt Mullenweg has said

By Tom Espiner | April 14, 2011 -- 13:31 GMT (06:31 PDT)

Follow @tomespiner

How can we prevent attackers from obtaining the data even if they gain access to the system?

The data should be encrypted at all times!

– Not just sometimes when the data is *at rest* (i.e. when no computations are being performed)

Problem: How does the server carry out its service (i.e. perform computations on the data) if the data is encrypted?

Two main approaches

1. Compute ***directly on encrypted data*** (uses specialized cryptography)

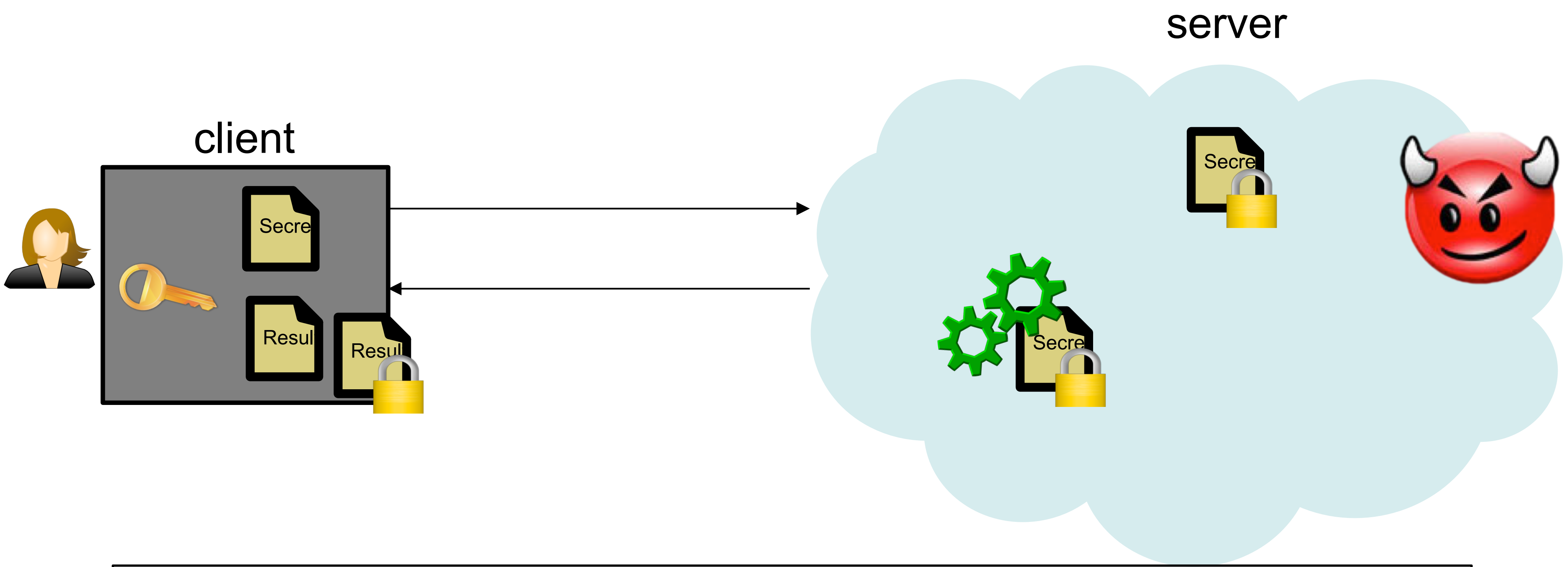
2. ***Shielded computation*** on data (uses specialized hardware)

Which approach to use?

– Security: confidentiality / integrity guarantees
– Functionality: what computations can be supported
– Performance: how efficient is it to compute

Approach #1: Computation on encrypted data

Computation on encrypted data



Server performs computations on the encrypted data without ever decrypting it

Computation on encrypted data

- Option #1: **Property preserving** encryption

Computation on encrypted data

- Option #1: **Property preserving** encryption
  - **Deterministic** encryption:

    If $\mathrm{x} = \mathrm{y}$ then $\mathrm{Enc(x)} = \mathrm{Enc(y)}$

Computation on encrypted data

- Option #1: **Property preserving** encryption
  - **Deterministic** encryption:

    If $x = y$ then $\mathrm{Enc}(x) = \mathrm{Enc}(y)$

Can compute queries such as:

```
SELECT * FROM table WHERE name = 'Alice'
```

Computation on encrypted data

- Option #1: **Property preserving** encryption
  - **Deterministic** encryption:

    If $x = y$ then $\mathrm{Enc}(x) = \mathrm{Enc}(y)$

Can compute queries such as:

```
SELECT * FROM table WHERE name = 0xfadc…
```

Computation on encrypted data

- Option #1: **Property preserving** encryption
  - Deterministic encryption:

    If $x = y$ then $\mathrm{Enc}(x) = \mathrm{Enc}(y)$

Can compute queries such as:

```
SELECT * FROM table WHERE name = 0xfadc…
```

  - **Order preserving** encryption:

    If $x > y$ then $\mathrm{Enc}(x) > \mathrm{Enc}(y)$

Computation on encrypted data

- Option #1: **Property preserving** encryption
  - Deterministic encryption:

    If $x = y$ then $\mathrm{Enc}(x) = \mathrm{Enc}(y)$

Can compute queries such as:
```
SELECT * FROM table WHERE name = 0xfadc...
```

  - **Order preserving** encryption:

    If $x > y$ then $\mathrm{Enc}(x) > \mathrm{Enc}(y)$

Can compute queries such as:
```
SELECT * FROM table WHERE age > 10
```

Computation on encrypted data

- Option #1: **Property preserving** encryption
  - Deterministic encryption:

    If $x = y$ then $\mathrm{Enc}(x) = \mathrm{Enc}(y)$

Can compute queries such as:

```
SELECT * FROM table WHERE name = 0xfadc…
```

  - **Order preserving** encryption:

    If $x > y$ then $\mathrm{Enc}(x) > \mathrm{Enc}(y)$

Can compute queries such as:

```
SELECT * FROM table WHERE age > 0x1d3e…
```

Computation on encrypted data

- Option #1: **Property preserving** encryption

Performance

- Nearly as fast as computing on plaintext

Security

- Leaks some information about the plaintexts (e.g. frequency distribution of values, or order of ciphertexts)

Functionality

- Very limited: e.g. only equality for deterministic encryption, range comparison for order preserving encryption

Computation on encrypted data

- Option #2: **Partially homomorphic** encryption

Computation on encrypted data

- Option #2: **Partially homomorphic** encryption
  - **ElGamal** cryptosystem (enables multiplication over ciphertexts)

$$\mathrm{Enc}(x) \,.\, \mathrm{Enc}(y) = \mathrm{Enc}(x \,.\, y)$$

Computation on encrypted data

- Option #2: **Partially homomorphic** encryption
  - **ElGamal** cryptosystem (enables multiplication over ciphertexts)

    $$\mathrm{Enc}(x)\,.\,\mathrm{Enc}(y) = \mathrm{Enc}(x\,.\,y)$$
  - **Paillier** cryptosystem (enables addition over ciphertexts)

    $$\mathrm{Enc}(x) + \mathrm{Enc}(y) = \mathrm{Enc}(x + y)$$

Computation on encrypted data

- Option #2: **Partially homomorphic** encryption

Performance

- Reasonably efficient, but not as fast as computing on plaintext

Security

- Similar level of confidentiality guarantees as standard AES-based encryption

Functionality

- Very limited: only specific operations can be computed (i.e. can only add, or can only multiply; can’t do both)

Computation on encrypted data

- Option #3: **Fully homomorphic** encryption
  - Enables ***arbitrary*** functions

$$F(\mathrm{Enc}(\mathrm{x}), \mathrm{Enc}(\mathrm{y})) = \mathrm{Enc}(F(\mathrm{x}, \mathrm{y}))$$

Computation on encrypted data

- Option #3: **Fully homomorphic** encryption
  - Enables ***arbitrary*** functions

$$F(\mathrm{Enc}(x), \mathrm{Enc}(y)) = \mathrm{Enc}(F(x, y))$$

Performance

- **Prohibitively slow** (currently 6 orders of magnitude slower)

Security

- Similar level of confidentiality guarantees as standard AES-based encryption

Functionality

- Allows arbitrary computations

Approach #2: Shielded computation on data using Intel Software Guard Extensions (SGX)

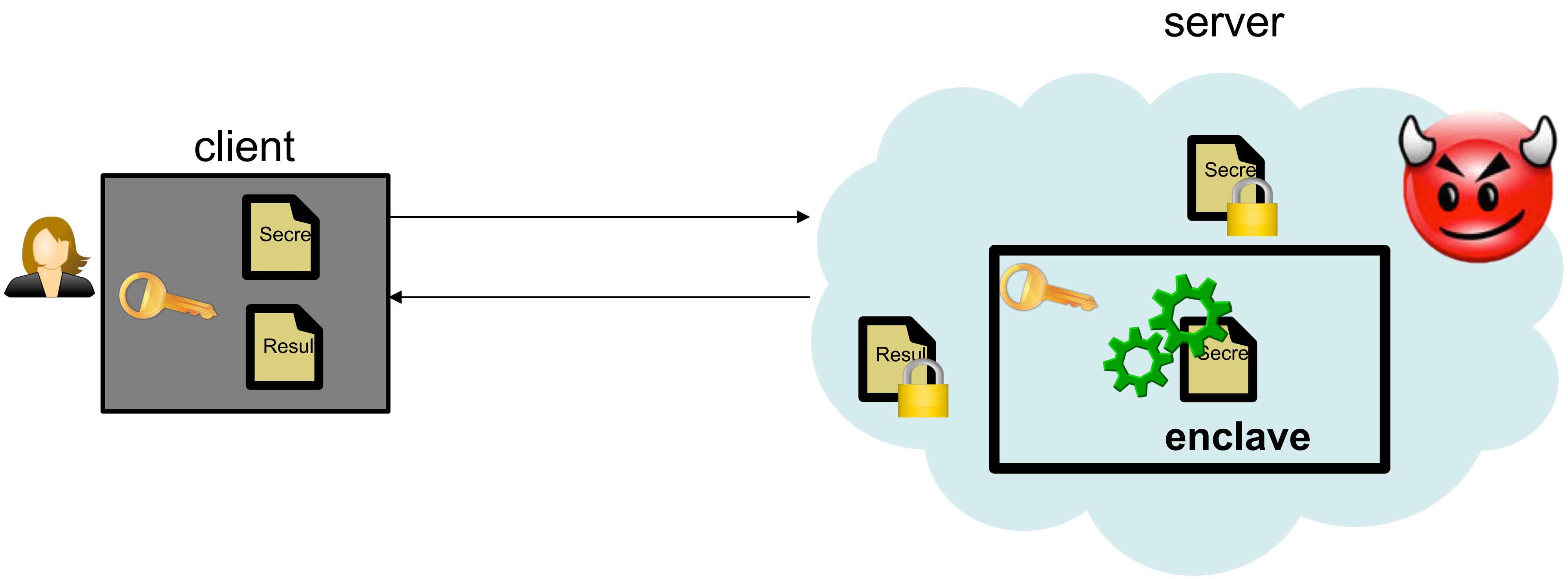
(Extensions to Intel processors)

Intel SGX

- Feature #1: Can run code in hardware-protected *containers* (called **enclaves**)

Intel SGX

- Feature #1: Can run code in hardware-protected *containers* (called **enclaves**)



Intel SGX

- Feature #1: Can run code in hardware-protected *containers* (called **enclaves**)
  - **Secure region of address space**, protected by the processor from all external software access (even from the operating system)

Intel SGX

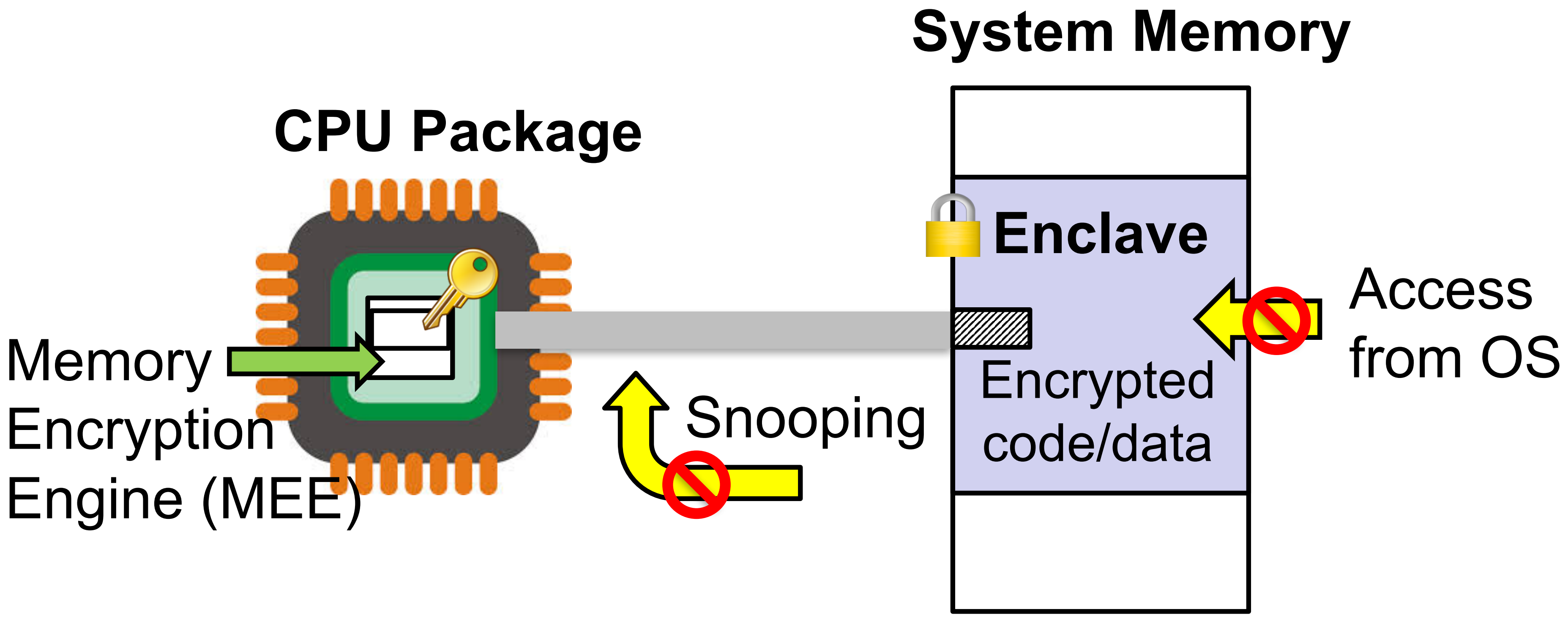
- Feature #1: Can run code in hardware-protected *containers* (called **enclaves**)
  - **Secure region of address space**, protected by the processor from all external software access (even from the operating system)
  - Code and data in enclave region of main memory **always encrypted** using **processor specific keys**
  - Decrypted only within the CPU package (i.e. when loaded into registers / cache)

Intel SGX

- Feature #1: Can run code in hardware-protected *containers* (called **enclaves**)
  - **Secure region of address space**, protected by the processor from all external software access (even from the operating system)
  - Code and data in enclave region of main memory **always encrypted** using **processor specific keys**
  - Decrypted only within the CPU package (i.e. when loaded into registers / cache)

Code and data loaded into an enclave is ***isolated*** from the rest of the system

SGX: How enclaves work



Problem: How to verify correct code has been loaded?

- Enclave code allowed to access unencrypted data
- Malicious / tampered code in enclave could *exfiltrate* data (i.e. leak it to the attacker)

Intel SGX

Extensions to Intel processors that support:

- Feature #2: **Attestation**

Intel SGX

- Feature #2: **Attestation**
  - Prove to local / remote system that the ***correct*** code has been loaded into the enclave (i.e. **verify the integrity of the enclave** using a **hash measurement** of the loaded code/data)

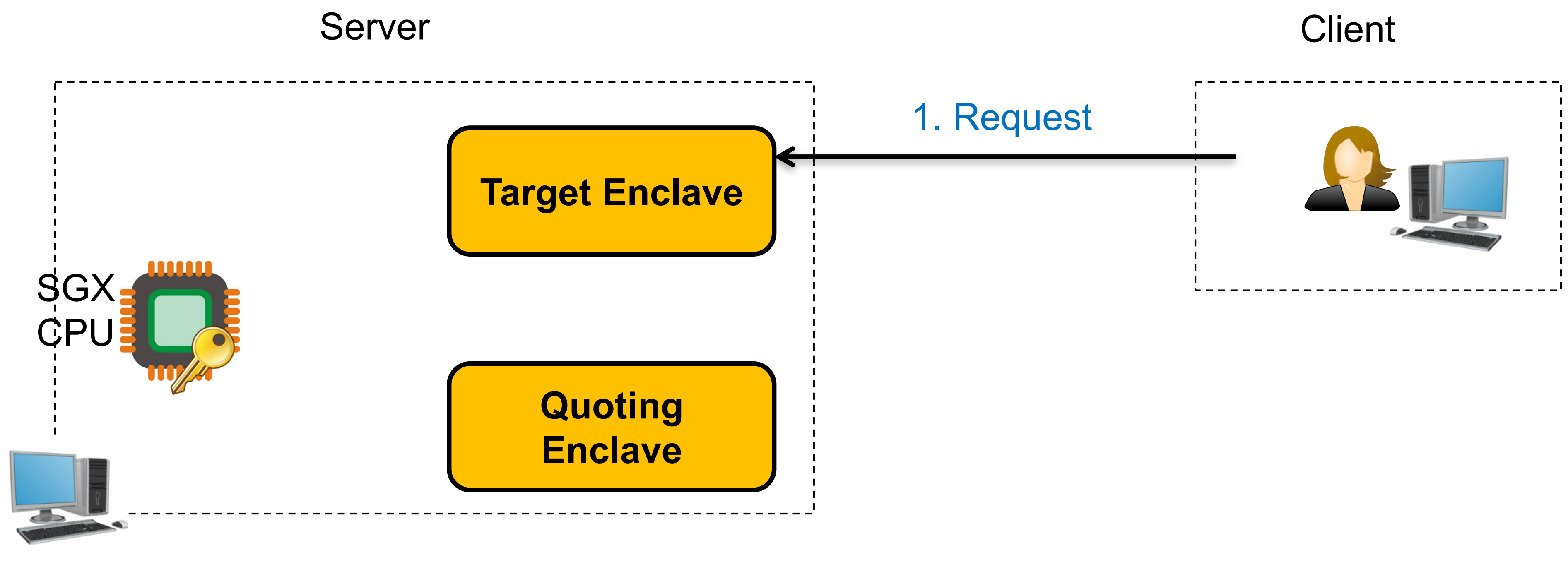
Intel SGX

- Feature #2: **Attestation**
  - Prove to local / remote system that the ***correct*** code has been loaded into the enclave (i.e. **verify the integrity of the enclave** using a **hash measurement** of the loaded code/data)
  - Verify that **measurement was generated by an enclave** running on the same platform (using a **MAC**)

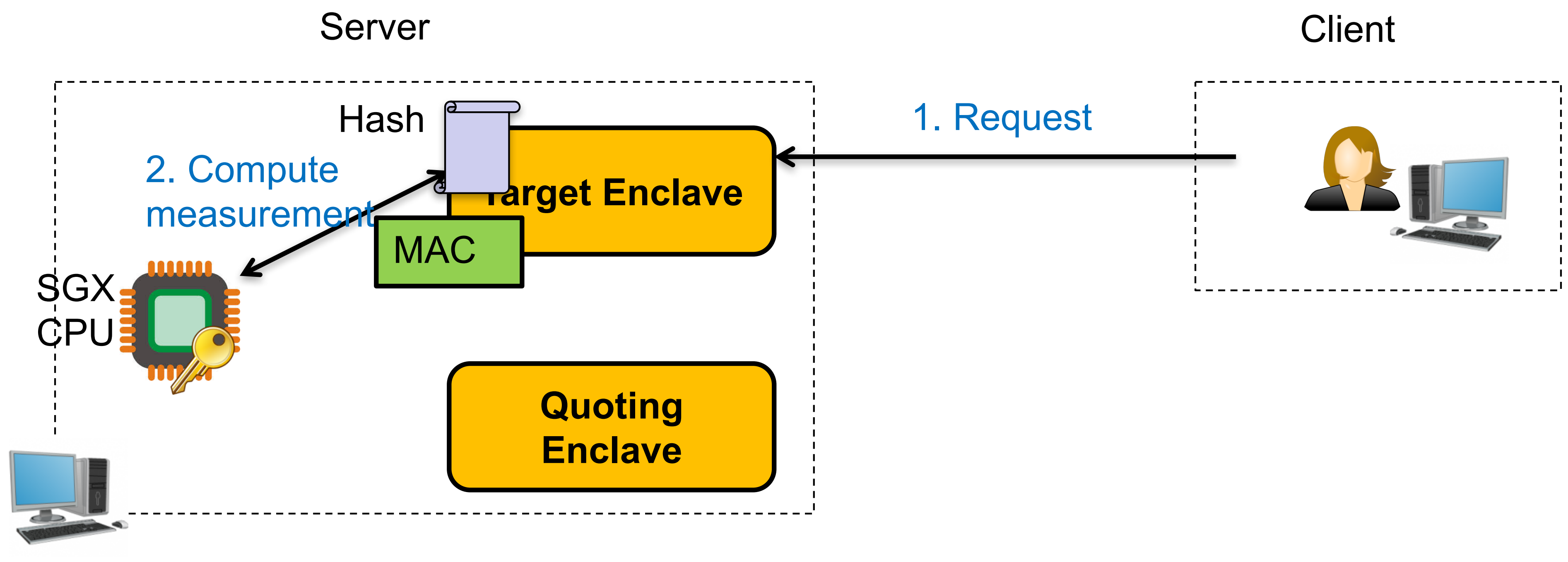
Intel SGX

- Feature #2: **Attestation**
  - Prove to local / remote system that the ***correct*** code has been loaded into the enclave (i.e. **verify the integrity of the enclave** using a **hash measurement** of the loaded code/data)
  - Verify that **measurement generated by an enclave** running on the same platform (using a **MAC**)
  - Uses a special ***quoting enclave*** for this purpose that signs the measurement and sends it to the client for verification

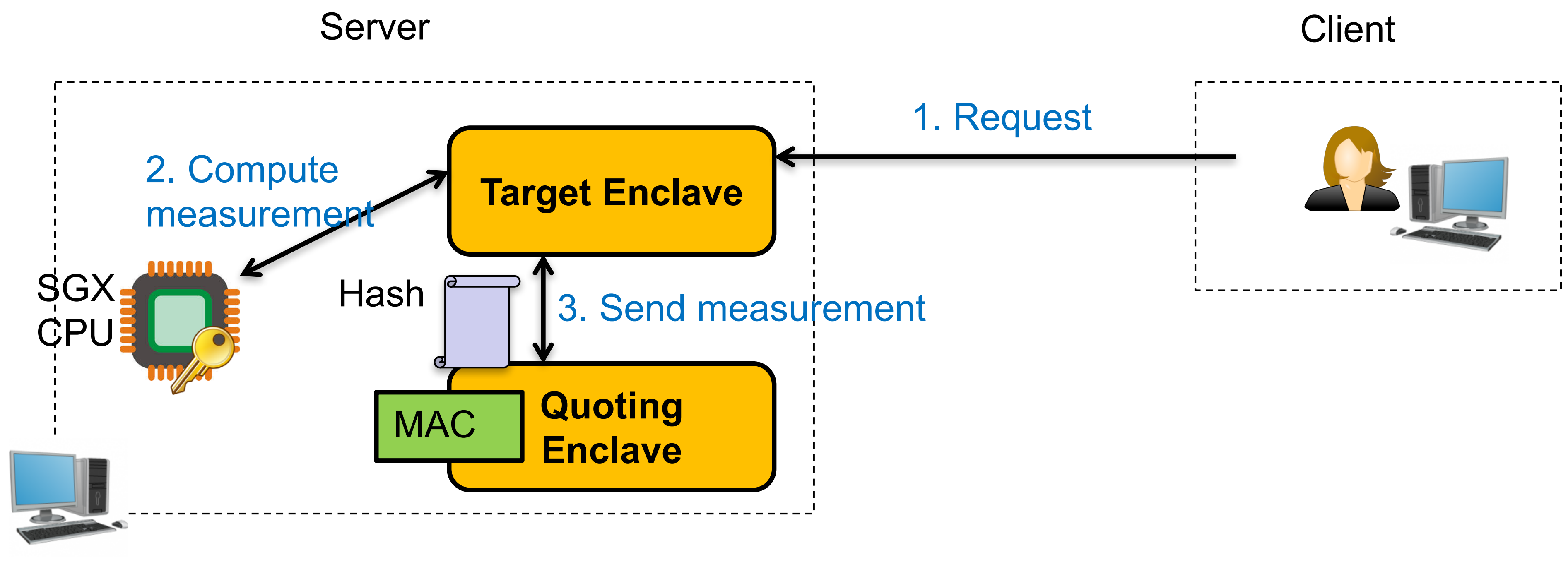
SGX: How attestation works



SGX: How attestation works

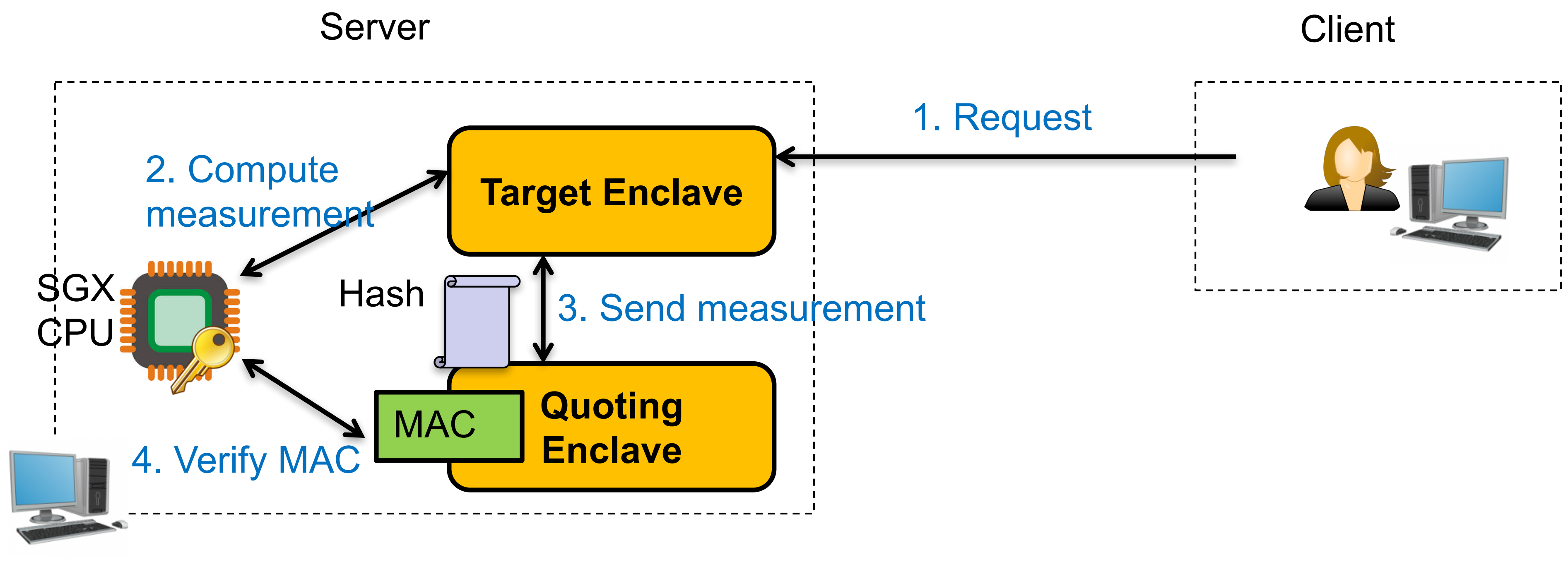


SGX: How attestation works



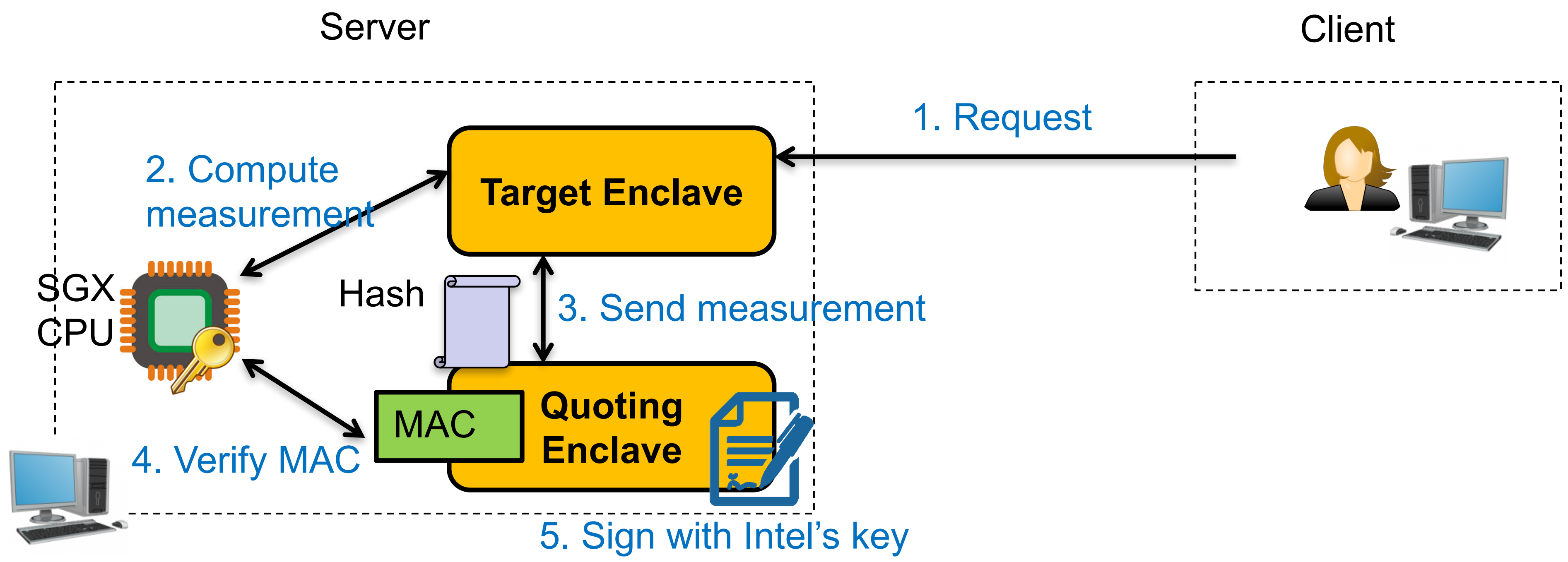
- Can also establish a **secure channel** between client and the enclave by exchanging Diffie-Hellman keys as part of the attestation process

SGX: How attestation works



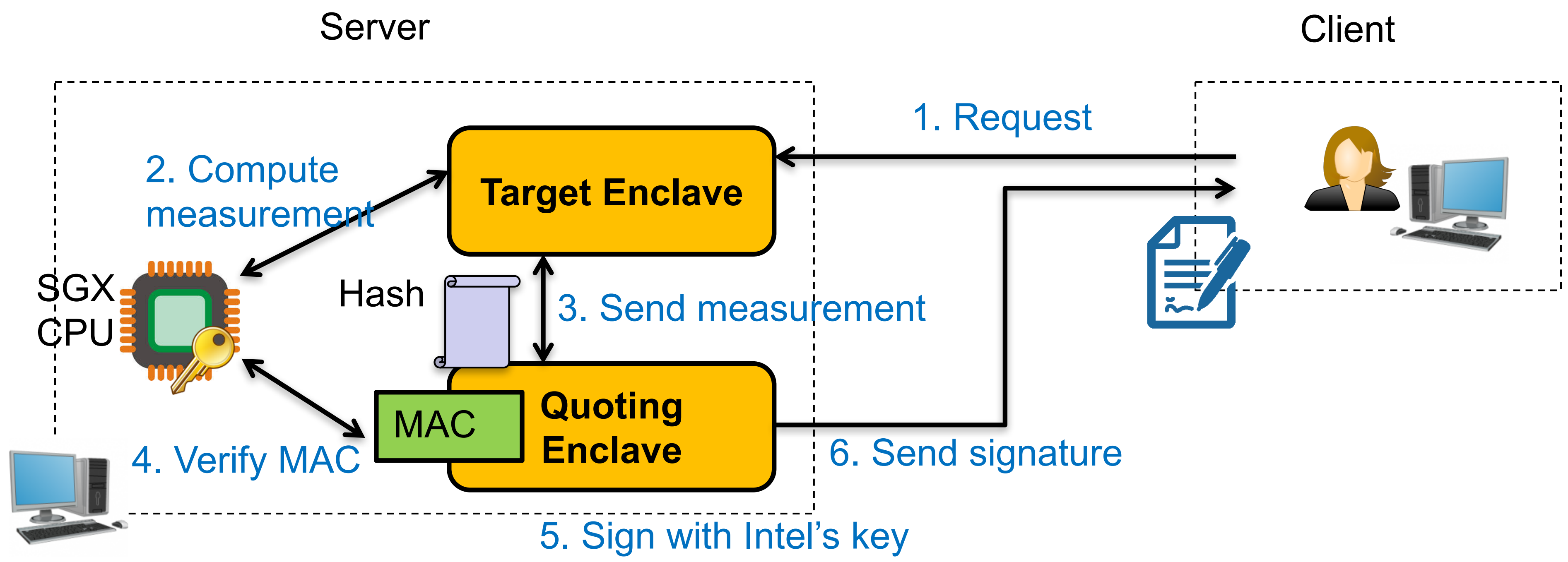
- Can also establish a **secure channel** between client and the enclave by exchanging Diffie-Hellman keys as part of the attestation process

SGX: How attestation works



- Can also establish a **secure channel** between client and the enclave by exchanging Diffie-Hellman keys as part of the attestation process

SGX: How attestation works



- Can also establish a **secure channel** between client and the enclave by exchanging Diffie-Hellman keys as part of the attestation process

Intel SGX

- **Minimal *TCB*** (trusted computing base)**:**
  - Only the **processor + the code loaded into the enclave** need to be trusted
  - Nothing else (DRAM, peripherals, operating system, etc.) needs to be trusted

Intel SGX

- **Minimal *TCB*** (trusted computing base)**:**
  - Only the **processor + the code loaded into the enclave** need to be trusted
  - Nothing else (DRAM, peripherals, operating system, etc.) needs to be trusted

So even if an attacker manages to gain root access on the server, won’t be able to learn the data

Summary

Comparison

Computation with cryptography	Computation with SGX
No need to trust server-side hardware	Need to trust Intel’s processor
No need to trust server-side software	Software running in enclave can leak unencrypted data No need to trust other privileged software (including the OS)
Can execute only a few simple functions efficiently	Runs arbitrary computation at processor speeds
Still vulnerable to side-channels	Still vulnerable to side-channels

Thank you, and good luck!