Announcements/Reminders

HW5 due Nov 10

PR2 due Dec 5
Previously on COS 433...
Digital Signatures
(aka public key MACs)
Message Integrity in Public Key Setting

Goal: If Eve changed \( m \), Bob should reject
Digital Signatures

Algorithms:

- Gen() \rightarrow (sk, pk)
- Sign(sk, m) \rightarrow \sigma
- Ver(pk, m, \sigma) \rightarrow 0/1

Correctness:

Pr[Ver(pk, m, Sign(sk, m))=1: (sk, pk)\leftarrow \text{Gen()}]=1
Building Digital Signatures

Non-trivial to construct with provable security

Most efficient constructions have heuristic security
Signatures from TDPs

\[ \text{Gen}_{\text{Sig}}() = \text{Gen}() \]

\[ \text{Sign}(sk, m) = F^{-1}(sk, H(m)) \]

\[ \text{Ver}(pk, m, \sigma): F(pk, \sigma) \overset{\equiv}{=} H(m) \]

**Theorem:** If \((\text{Gen}, F, F^{-1})\) is a secure TDP, and \(H\) is “modeled as a random oracle”, then \((\text{Gen}_{\text{Sig}}, \text{Sign}, \text{Ver})\) is (strongly) CMA-secure
Basic Rabin Signatures

\[ \text{Gen}_{\text{Sig}}() : \] let \( p, q \) be random large primes
\[ \text{sk} = (p, q), \text{pk} = N = pq \]

\[ \text{Sign}(\text{sk}, m) : \text{Solve equation} \sigma^2 = H(m) \mod N \text{ using factors} p, q \]
\[ \text{• Output} \sigma \]

\[ \text{Ver}(\text{pk}, m, \sigma) : \sigma^2 \mod N = H(m) \]
Today

Signatures cont.
Identification protocols
Schnorr Signatures

sk = w
pk = h:=g^w

Sign(sk,m):
• r←\mathbb{Z}_p
• a←g^r
• b←H(m,a)
• c←r+wb
• Output (a,c)

Ver(h,m,(a,c)):
• b←H(m,a)
• a\times h^b =? g^c?

Theorem: If Dlog is hard and H is modeled as a random oracle, then Schnorr signatures are strongly CMA secure
What’s the Smallest Signature?

RSA Hash-and-Sign: 2 kilobits

ECDSA (variant of Schnorr using “elliptic curves”): around 512 bits

BLS: 256 bits

Are 128-bit signatures possible?
- No fundamental reason for impossibility, but all (practical) schemes require 256 bits or more
Digital Signatures and the Public Key Infrastructure
Digital Signatures and the Public Key Infrastructure
Digital Signatures and the Public Key Infrastructure

\[ c' = \text{Enc}(pk', m) \]
Digital Signatures and the Public Key Infrastructure

\[ c' = \text{Enc}(pk', m) \]

\[ c = \text{Enc}(pk, m) \]
Takeaway

Need some authenticated channel to ensure distribution of public keys

But how to authenticate channel in the first place without being able to distribute public keys?
Solution: Certificate Authorities

CA

$sk_{CA}$

Business
Government Agency
Department within company

$pk_{CA}$
Solution: Certificate Authorities

\[ \text{Cert}_{CA \rightarrow B} = \text{Sign}(sk_{CA}, \text{"Bob’s public key is pk}_B\text{"}) \]
Solution: Certificate Authorities

Bob is typically some website
- Obtains $\text{Cert}$ by, say, sending someone in person to CA with $\text{pk}_B$
- Only needs to be done once

If Alice trusts CA, then Alice will be convinced that $\text{pk}_B$ belongs to Bob

Alice typically gets $\text{pk}_{\text{CA}}$ bundled in browser
Limitations

Everyone must trust same CA
• May have different standards for issuing certs

Single point of failure: if $sk_{CA}$ is compromised, whole system is compromised

Single CA must handle all verification
Multiple CAs

There are actually many CA’s, CA₁, CA₂,...

Bob obtains cert from all of them, sends all the certs with his public key

As long as Alice trusts one of the CA’s, she will be convinced about Bob’s public key
Certificate Chaining

CA issues $\text{Cert}_{\text{CA} \rightarrow \text{B}}$ for Bob

Bob can now use his signing key to issue $\text{Cert}_{\text{B} \rightarrow \text{D}}$ to Donald

Donald can now prove his public key by sending $(\text{Cert}_{\text{CA} \rightarrow \text{B}}, \text{Cert}_{\text{B} \rightarrow \text{D}})$
• Proves that CA authenticated Bob, and Bob authenticated Donald
Certificate Chaining

For Bob to issue his own certificates, a standard cert should be insufficient
• CA knows who Bob is, but does not trust him to issue certs on its behalf

Therefore, Bob should have a stronger cert:

\[ \text{Cert}_{\text{CA} \rightarrow \text{B}} = \text{Sign}(\text{sk}_\text{CA}, \text{“Bob’s public key is } \text{pk}_\text{B} \text{ and he can issue certificates on behalf of CA”}) \]
Certificate Chaining

One root CA

Many second level CAs $\text{CA}_1$, $\text{CA}_2$, ...

- Each has $\text{cert}_{\text{CA} \rightarrow \text{CA}_i}$

Advantage: eases burden on root

Disadvantage: now multiple points of failure
Invalidating Certificates

Sometimes, need to invalidate certificates
• Private key stolen
• User leaves company
• Etc

Options:
• Expiration
• Explicit revocation
Identification Protocols
Identification
Identification
Identification

To identify yourself, you need something the adversary doesn’t have

Typical factors:
• What you are: biometrics (fingerprints, iris scans,...)
• What you have: Smart cards, SIM cards, etc
• What you know: Passwords, PINs, secret keys
Types of Identification Protocols

Secret key:

Public Key:
Types of Attacks

Direct Attack:
Types of Attacks

Eavesdropping/passive:

\(sk\) \(\rightarrow\) \(vk\)
Types of Attacks

Eavesdropping/passive:

sk

vk
Types of Attacks

Man-in-the-Middle/Active:

sk ———> vk
Types of Attacks

Man-in-the-Middle/Active:

\[ \text{sk} \leftrightarrow \text{Alice} \leftrightarrow \text{Robot} \leftrightarrow \text{vk} \]
Basic Password Protocol

Never ever (ever ever...) use

\[ sk = \text{pwd} \quad \text{and} \quad vk = \text{pwd} \]

Is \( sk \) equal to \( vk \)?
Problem with Basic Pwd Protocol

vk must be kept secret at all costs

Issue:

<table>
<thead>
<tr>
<th>User</th>
<th>Pwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>pwd_A</td>
</tr>
<tr>
<td>Bob</td>
<td>pwd_B</td>
</tr>
<tr>
<td>Charlie</td>
<td>pwd_C</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Problem with Basic Pwd Protocol

vk must be kept secret at all costs

Issue:
Slightly Better Version
STILL never ever (ever ever...) use

Let $H$ be a hash function

$sk = pwd$

$vk = H(pwd)$

$H(sk) == vk?$
Slightly Better Version
STILL never ever (ever ever...) use

Let $H$ be a hash function

<table>
<thead>
<tr>
<th>User</th>
<th>Pwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$H(\text{pwd}_A)$</td>
</tr>
<tr>
<td>Bob</td>
<td>$H(\text{pwd}_B)$</td>
</tr>
<tr>
<td>Charlie</td>
<td>$H(\text{pwd}_C)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Slightly Better Version

STILL never ever (ever ever...) use

Advantage of hashing:
• Now if pwd database is leaks, adversary only gets hashes passwords

• For identification protocol, need actual password

• Therefore, adversary needs to invert hash function to break protocol

• Presumed hard
Weak Passwords

Data from 10M passwords leaked in 2016:

<table>
<thead>
<tr>
<th>RANK</th>
<th>PASSWORD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>123456</td>
</tr>
<tr>
<td>2.</td>
<td>123456789</td>
</tr>
<tr>
<td>3.</td>
<td>qwerty</td>
</tr>
<tr>
<td>4.</td>
<td>12345678</td>
</tr>
<tr>
<td>5.</td>
<td>111111</td>
</tr>
<tr>
<td>6.</td>
<td>1234567890</td>
</tr>
<tr>
<td>7.</td>
<td>1234567</td>
</tr>
<tr>
<td>8.</td>
<td>password</td>
</tr>
<tr>
<td>9.</td>
<td>123123</td>
</tr>
<tr>
<td>10.</td>
<td>987654321</td>
</tr>
<tr>
<td>11.</td>
<td>qwertyuiop</td>
</tr>
<tr>
<td>12.</td>
<td>mynoob</td>
</tr>
<tr>
<td>13.</td>
<td>123321</td>
</tr>
<tr>
<td>14.</td>
<td>666666</td>
</tr>
<tr>
<td>15.</td>
<td>18atcskd2w</td>
</tr>
<tr>
<td>16.</td>
<td>7777777</td>
</tr>
<tr>
<td>17.</td>
<td>1q2w3e4r</td>
</tr>
<tr>
<td>18.</td>
<td>654321</td>
</tr>
<tr>
<td>19.</td>
<td>555555</td>
</tr>
<tr>
<td>20.</td>
<td>3rjs1ia7qe</td>
</tr>
<tr>
<td>21.</td>
<td>google</td>
</tr>
<tr>
<td>22.</td>
<td>1q2w3e4r5t</td>
</tr>
<tr>
<td>23.</td>
<td>123qwe</td>
</tr>
<tr>
<td>24.</td>
<td>zxcvbnm</td>
</tr>
<tr>
<td>25.</td>
<td>1q2w3e</td>
</tr>
</tbody>
</table>

50% of available passwords

https://blog.keepersecurity.com/2017/01/13/most-common-passwords-of-2016-research-study/
Weak Passwords

Of course, pwds that have been leaked are likely the particularly common ones

Even so, 360M pwds covers about 25% of all users
Online Dictionary Attacks

Suppose attacker gets list of usernames

Attacker tries logging in to each with $pwd = '123456'$

5-17% of accounts will be compromised
Online Dictionary Attacks

How to slow down attacker?
• Lock out after several unsuccessful attempts
  • Honest users may get locked out too

• Slow down response after each unsuccessful attempt
  • 1s after 1\textsuperscript{st}, 2s after 2\textsuperscript{nd}, 4s after 3\textsuperscript{rd}, etc
Offline Dictionary Attacks

Suppose attacker gets hashed password $vk = H(pwd)$

Attack:
• Assemble dictionary of 360M common passwords
• Hash each, and check if you get $vk$
• If so, you have just found $pwd$!

On modern hardware, takes a few seconds to recover a passwords 25% of the time
Offline Dictionary Attacks

Now consider what happens when adversary gets entire hashed password database

- Hash dictionary once: $O(|D|)$
- Index dictionary by hashes
- Lookup each database entry in dictionary: $O(|L|)$

To get 25% of passwords takes $O(|D|+|L|)$ time
- Amortize cost of hashing dictionary over many passwords
Salting

Let $H$ be a hash function

$s_i$ random

<table>
<thead>
<tr>
<th>User</th>
<th>Salt</th>
<th>Pwd</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>$s_A$</td>
<td>$H(s_A, pwd_A)$</td>
</tr>
<tr>
<td>Bob</td>
<td>$s_B$</td>
<td>$H(s_B, pwd_B)$</td>
</tr>
<tr>
<td>Charlie</td>
<td>$s_C$</td>
<td>$H(s_C, pwd_C)$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Salting

Salt length? Enough to make each user’s salt unique
• At least 64 bits

Salting kills amortization:
• To recover Alice’s key, adversary must hash entire dictionary with $s_A$
• To recover Bob’s key, adversary must hash entire dictionary with $s_B$
• Must hash entire dictionary again for each user

Running time: $O(|D| \times |L|)$
Unique Passwords

Different websites may employ different standards for password security
• Some may store passwords in clear, some may hash without salt, some may salt

If you use the same password at a bank (high security) and your high school reunion (low security), could end up with your password stolen
Unique Passwords

Solutions:
• Password managers

• Salt master password to generate website-specific password (e.g. pwdhash):

  Master password: pwd
  Pwd for abcdefg.com: H(abcdefg.com,pwd)
What Hash Function to Use

In LindedIn leak (using Sha1), 90% of passwords were recovered within a week

Problem: Sha1 is very fast!

To make hashing harder, want hash function that is just slow enough to be unnoticeable to user
What Hash Function to Use

Examples: PBKDF2, bcrypt

• Iterate hash function many times:

\[ H'(x) = H(H(H(\ldots H(x) \ldots))) \]

• Set #iterations to get desired hashing time

Still problem:

• Adversary may have special purpose hardware

\[ \Rightarrow \text{Can eval much faster than you can (50,000x)} \]
What Hash Function to Use

Memory-hard functions: functions that require a lot of memory to compute
• As far as we know, no special purpose memory
• Attacker doesn’t gain advantage using special purpose hardware

Examples: Scrypt, Argon2i