DIGITAL SIGNATURES AND CERTIFICATES

NOTICES

› Homework #3 is out, due 23:59 Tuesday, Feb 12 (1 week)
› Lab 2 due 23:59 Thursday Feb. 15 (~1 week)

DIGITAL SIGNATURES AND CERTIFICATES

CRYPTO RECAP

› Symmetric key encryption
  › One-time pads
  › Block ciphers
  › Stream ciphers
› Message Authentication Codes
  › Secure hash functions
  › Unforgeability
› Authenticated encryption
› Public key exchange
› Public-key encryption

DIGITAL SIGNATURES AND CERTIFICATES

TODAY

› Digital signatures
› Secure Key Exchange
› Digital Certificates
› Key Distribution
› HW#2 review

Cryptography : Goals

• Confidentiality
  – Preventing adversaries from reading our private data
• Integrity
  – Preventing attackers from altering our data
  – Data itself might not be confidential
• Authentication
  – Determining who created a given message or document
  – Generally implies/requires integrity

Cryptography : The Players

• Alice ⬆️ Sender of messages

• Bob ⬇️ Receiver of messages

• Eve ⬇️ Eavesdropper (interested in reading message content)

• Mallory 😈 Manipulator (interested in altering message content)
Digitized Signatures

- **Goal**: Demonstrate that signee produced/endorsed the document

- **Problem**: Attacker can copy Alice’s digitized signature from one document to another – just as they could forge a written one

Digital Signatures

- **Solution**: Make signature $S$ document specific

\[ S = F(M, Alice) \]

- Given signature $S$ and the document:
  - Need to be able to confirm that only Alice could have produced $S$ using some verification function $V(M, Alice)$
  - Discard as forgery (or corrupted) if not

RSA Digital Signatures

- Alice generates public/private key pair, $(n', e')$ and $(d')$
  (advisedly ≠ her public/private keys for encryption!)
- Then chooses and makes public a secure hash function $H$
- To sign a message $M$, Alice computes:
  \[ S = \text{SIGN}_{d'}(H(M)) = H(M)^{d'} \mod n' \]
- Anyone (not just recipient) can verify her signature on $(M, S)$ via:
  \[ \text{VERIFY}_{n', e'}(M, S) = \text{true} \text{ if and only if } H(M) = S^{e'} \mod n' \]
- This follows from:
  \[ (H(M)^{d'})^{e'} = (H(M)^e)^{d'} = H(M) \mod n' \]
  (by previous analysis of RSA)

Digital Signatures Considerations

- Any change to $M$ will alter $H(M)$, and hence the computed $S$ ⇒ detectable ⇒ integrity
- Security rests on the difficulty of finding the inverse of $e$, along with $H$ being cryptographically strong
- Anyone can confirm signature is valid if Alice’s public signature key is well-known ⇒ non-repudiation
- **Non-repudiation**:
  - Alice can’t deny to a third party that she signed $M$
    - Unless she claims her private key was stolen
  - Similar to a handwritten signature, but better since it can’t be “digitized” and pasted into another document $M^*$ because $(M^*, S)$ won’t validate

Agreeing on Secret Keys

Without Prior Arrangement
Diffie-Hellman Key Exchange

- While we have powerful symmetric-key technology, it requires Alice & Bob to agree on a secret key ahead of time.
- Would be easier if they can generate a key only when needed.
- But how can they exchange such a key without Eve learning it?
- Possible using public-key technology:
  - Requires that Alice & Bob know that their messages will reach one another without any meddling.
  - So works against Eve (an eavesdropper), but not Mallory (a man-in-the-middle attacker).
  - Protocol: Diffie-Hellman Key Exchange (DHE)

1. Bob and Alice agree in advance on a well-known (large) prime $p$ and a corresponding $g \in (1, \ldots, p-1)$ both of which Eve will overhear.

2. Alice picks random secret $a\in(1, \ldots, p-1)$.

3. Bob picks random secret $b\in(1, \ldots, p-1)$.

4. Alice sends $A = g^a \mod p$ to Bob.

5. Bob sends $B = g^b \mod p$ to Alice.

6. Alice knows $(a, A, B)$, computes $K = B^a \mod p = (g^b)^a = g^{ab} \mod p$.

7. Bob knows $(b, A, B)$, computes $K = A^b \mod p = (g^a)^b = g^{ab} \mod p$.

8. $K$ is now the shared secret key.

While Eve knows $(p, g, g^a \mod p, g^b \mod p)$, it is computationally infeasible for her to then deduce $K = g^{ab} \mod p$ (at least, believed to be infeasible).

She can easily construct $A\cdot B = g^a g^b \mod p = g^{a+b} \mod p$, but computing $g^{ab}$ requires the ability to take discrete logarithms $\mod p$ (so, so hard 😞).
1. Bob and Alice agree in advance on a well-known (large) prime $p$ and a corresponding $g$: $1 < g < p - 1$

But what happens if instead of Eve watching, Alice & Bob face the threat of a hidden Mallory (MITM) attack?

2. Alice picks random secret $a$: $1 < a < p - 1$

3. Bob picks random secret $b$: $1 < b < p - 1$

4. Alice sends $A = g^a \mod p$ to Bob

5. Mallory prevents Bob from receiving $A$

6. Mallory generates a new $a'$ and $b'$

7. Mallory sends $A' = g^{a'} \mod p$ to Bob

8. Same happens for Bob and $B/B'$
Attacking DHE

Alice
\[ p, g, a \]

Mallory
\[ p, g, a', b' \]

Bob
\[ p, g, b \]

\[ A = g^a \mod p \quad A' = g^{a'} \mod p \]
\[ B = g^b \mod p \quad B' = g^{b'} \mod p \]
\[ K_s = (B)^{a'} \mod p = (g^b)^{a'} = g^{ab} \mod p \]
\[ K_s' = (A')^b \mod p = (g^{a'})^b = g^{a'b} \mod p \]

9. Alice and Bob now compute keys they share with Mallory!

10. Mallory can relay encrypted traffic between the two... modifying it or injecting new data as desired 😊

Public Key Distribution

For simplicity, assume Alice uses the same key for encryption & signing
Communicating with Someone New

- **Public-key Cryptography** gives us astonishing capabilities to achieve confidentiality, integrity, and authentication without shared secrets.
- But how do we avoid MITM attacks?
- How can we trust we have the true public key for someone we want to communicate with?
- Any Ideas?

Trusted Authorities

- Suppose there’s a party that everyone agrees to trust to confirm every individual’s public key
  - Say... Her Majesty The Queen
    (she seems like a nice, trustworthy lady)
- Issues with this approach?
  - How can everyone agree to trust them?
  - Huge amount of work
  - Single point of failure
  - And thus Denial-of-Service concerns
  - How do you know you’re even talking to the right authority?
  - Maybe there’s an imposter, pretending to be HRH Queen Elizabeth II

Trust Anchors

- Suppose the trusted party distributes their key so everyone has it
- We can then use this to bootstrap trust
  - As long as we have confidence in the decisions that party makes

Digital Certificates

- A certificate ("cert") is a signed claim about someone’s key
- More broadly: a signed attestation about some claim
- Notation:
  \[ \{ M \}_{k} = \text{“message } M \text{ encrypted with public key } k” \]
  \[ \{ M \}_{k_{priv}} = \text{“message } M \text{ signed with private key for } K” \]
- Example:
  \[ M = \text{“Kate’s public key } K_{\text{pub}} \text{ is 0xF32A99B...”} \]
  Cert: \[ \{ \text{“Kate’s public key } K_{\text{pub}} \text{ is 0xF32A99B...”} \}_{K_{\text{priv}}} = 0x923AB9... \]
Her Royal Britannic Majesty hereby asserts:
Kate's public key $K_{Kate}$ is: 0xF32A99B...
The signature for this assertion using $K_{Queenie}$ is: 0x923AB95...

Her Royal Britannic Majesty hereby asserts:
Kate's public key $K_{Kate}$ is: 0xF32A99B...
The signature for this assertion using $K_{Queenie}$ is: 0x923AB95...

This... The signature for this assertion using $K_{Queenie}$ is: 0x923AB95...
And can be validated using this...

If We Find This Cert...
- What can we figure out?
  - If we know Queenie's key, then we can determine whether she did indeed sign the statement.
  - If we trust Queenie's decisions, then we have confidence we really have Kate's key.
- How do we determine trust?
  - We trust Queenie won't sign such statements on a whim.
  - We trust Kate won't let her private key be stolen.

Analysing Found Certs
- How we get the cert doesn't affect its utility at all.
- Who gave us the cert doesn't matter.
  - That person is no more or less trustworthy simply because they did.
- Possession of a cert does NOT establish identity.
  - However:
    - If someone demonstrates they can decrypt data encrypted with $K_{Kate}$, then we have high confidence they possess $K_{Kate}$.
    - Same if they show they can sign data using what appears to be $K_{Kate}$.

Scaling Digital Certificates
- How can this possibly scale? Surely HRH Queenie can't sign everyone's public key!
- Approach No. 1: Introduce hierarchy via delegation...
  - "Prince Phillip's public key is 0x... and I trust him to vouch for The United Kingdom" $K_{Queenie}$
  - "Prince William's public key is 0x... and I trust him to vouch for England" $K_{Queenie}$
  - "Prince George's public key is 0x... and I trust him to vouch that he did his homework" $K_{Queenie}$
  - "Princess Charlotte's public key is 0x... and I trust her to vouch that she ate her lunch" $K_{Queenie}$

Scaling Digital Certificates Cont.
- Say Charlotte puts this last Cert on her official web page.
  - Anyone can gather the intermediary keys and validate the chain.
  - They can get these (other than Queenie's) from anywhere because they can validate them, too.
- Approach No. 2: Have multiple trusted parties who are in the business of signing certs.
  - The certs might also be hierarchical as in Approach No. 1.
Certificate Authorities

- CAs are **trusted parties** in a **Public Key Infrastructure** (PKI)
- They typically operate offline
  - They sign ("cut") certs when convenient, not on-the-fly
- Suppose Alice wants to communicate confidentially with Bob:
  - Bob gets a CA to issue (Bob’s public key is $B) K_{CA}$
  - Alice gets Bob’s cert any old way
    - Via e-mail, from a website, on a USB stick found in the parking lot, shoved under the door...
  - Alice uses her known value of $K_{CA}$ to verify cert’s signature
  - Alice extracts $B$, sends $\{M\}_B$ to Bob
Certificate Revocation

- But what do we do if a CA messes up and issues a cert in Bob’s name to Mallory?
  - E.g. Verisign once issued a microsoft.com cert to a regular guy!
  - Related problem: Bob realizes b (his private key) has been stolen
- How do we recover from the error/loss?
- Approach No. 1: Expiration dates
  - Mitigates possible damage
  - But adds management burden:
    - Benign failures to renew will break normal operation
    - e.g. Staff member responsible for certs is on vacation when renewal comes up
  - Security Principle? Fail-safe Defaults

Certificate Revocation Cont.

- Approach No. 2: Announce revoked certs
  - Users periodically download a certificate revocation list (CRL)
- Issues?
  - Lists can get large
  - Need to authenticate the list itself — how? Sign it!
  - Mallory can exploit download lag
  - What does Alice do if she can’t reach CA for download?
    - Assume all certs are invalid:
      - Security Principle? Fail-safe Defaults
      - But wow, what an inconvenient failure mode!
    - Use old list:
      - Widens exploitation window if Mallory can mount a Denial of Service (DoS) attack on the CA preventing them from distributing a new CRL

Certificate Revocation Cont.

- Approach No. 3: CA provides service to query
  - OCSP: Online Certificate Status Protocol
- Issues?
  - Can’t be used if Alice doesn’t have connectivity to CA
  - CA learns that Alice talks to Bob (i.e. privacy concerns)
  - CA had better build this in a scalable fashion!
  - CA outages ⇒ big headaches
- OR: Alice defaults to trusting if OCSP inaccessible
  - Again, creates a DoS threat
  - Breaks Security Principles “Fail-safe Defaults”, “Default Deny”, etc. etc.

Leap-of-Faith Authentication

- A radically different approach that leverages key continuity
  - Also called Trust On First Use (TOFU)
  - Requires cert to have specific properties, like a particular CA
  - Very popular for SSH
  - Web browsers don’t expose an easy equivalent usage model
Leap-of-Faith Authentication

- A radically different approach that leverages key continuity
  - Also called Trust On First Use (TOFU)
  - Requires cert to have specific properties, like a particular CA
  - Very popular for SSH
    - Web browsers don’t expose an easy equivalent usage model
- Properties/Issues?
  - Doesn’t bug you, just automatically gives you a secure mode of operation
  - Great design property!
  - Leverages mental expectations
    - Such as: “hard for attacker to anticipate this’ll be my very first visit”
    - Clearly not always true!
  - Or: “Bob mentioned he’d be upgrading, so the new key is expected”
    - Brittle: Relies on user to notice and thoughtfully respond to key changes

Final Thoughts on Cryptography

- Security should be **good enough** for your threat model
- TOFU may well be sufficient for your everyday needs
- But continually re-evaluate your threat model
  - Increase security if necessary
  - Move to stronger encryption if necessary
- If “good enough” in your situation ⇒ super secure...
  - Insist on Fail-safe Defaults
  - Perhaps even write all your own code or have Ken Thompson write it for you 😊
  - But always Defend in Depth:
    - Use multiple CAs
    - Use a different cryptography scheme for every transmission type
    - Change them around at non predictable intervals
    - Employ multiple levels of encryption, encrypt cipher text with a different scheme