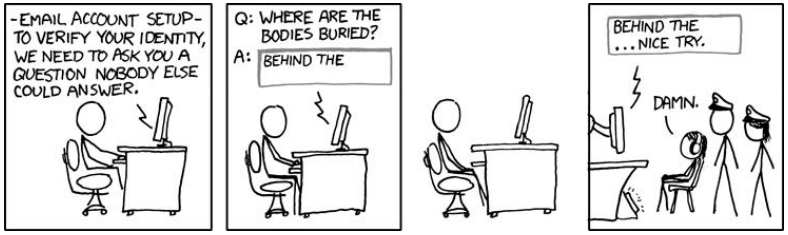
CPSC 418/MATH 318 Introduction to Cryptography

Cryptography in Practice: Key Management, Entity Authentication, a Real-World Solution (SSH)

Renate Scheidler

Department of Mathematics & Statistics
Department of Computer Science
University of Calgary

Week 12



Outline

Authentication

Today, *authentication* is arguably the most important application of cryptography. Three main classifications:

- Message authentication (MACs) — covered
- Data-origin authentication (digital signatures) — covered previously
- Authenticated key establishment — covered next
- Entity authentication (client-server, user-host, process-host) — covered after that

In practice, these are often combined into one protocol (e.g. SSL/TLS).

Authenticity of Keys

Secure communication requires proper mechanisms for managing keys and ensuring their **authenticity**.

Mechanisms for ensuring authenticity of keys:

- A trusted third party
  - A key distribution center (session keys)
  - A certification authority (public keys)
- Identity-based cryptography: your ID is your public key. A trusted authority derives users' private keys (and thus knows *all* private keys!)
- Peer authentication via a *web of trust* that establish the authenticity of the *binding* between a public key and its owner (Phil Zimmerman, 1992, used in PGP secure e-mail)

The vast majority of key distribution systems involve a **trusted authority** to ensure authenticity of keys.

Symmetric Key Distribution

Symmetric schemes require both parties to share a common, secret key.

Possible distribution mechanisms:

- A selects a key and physically delivers to B. Secure, but cumbersome.
- Third party selects and physically delivers key to A and B. Also secure, but cumbersome.
- A and B can use a previous key to encrypt a new key. If one key is compromised, all subsequent keys are compromised.
- A commonly-trusted third party called a *key distribution center* (KDC) can relay the key between A and B via encrypted links (commonly used solution).

Key Distribution Centres

Idea:

- Each user holds a shared symmetric **master key** with the KDC
- Master keys are used for distributing one-time **session keys**:
  - KDC generates a session key K shared between A and B
  - KDC encrypts K with the master key it shares with A and sends encryption to A
  - KDC encrypts K with the master key it shares with B and sends encryption to B
- A and B communicate using session key K for encryption and destroy K at the end of their session

Advantages:

- Far fewer long-term keys than if each pair of entities holds a shared long-term key
- Compromise of a session key does not affect master key nor other sessions

Key Distribution Centres: Issues

Issues:

- Hierarchies of KDC's required for large networks, must trust each other
- Session key lifetimes should be limited for greater security
- All keys and entities (users and KDCs) must be authenticated (more later)

Public Key Solutions

Key management in conventional cryptography is handled via *key distribution centres*. Now we look at public key solutions.

There are three main contributions in PKC:

- **Digital signatures** — for data origin authentication and non-repudiation
- **Key agreement protocols** — both parties contribute to the generation of a session key (eg. Diffie-Hellman)
- **Key transport** via hybrid encryption — party A generates a session key, encrypts and sends to B using a PKC (B has no control over the session key)

Main problem — user's public keys must be *authenticated* in order to prevent active attacks such as man-in-the-middle and impersonation.

Public-Key Distribution, I

1. **Point-to-point delivery over a trusted channel** such as personal exchange, registered mail, courier, etc.
   Problems: slow, inconvenient, potentially expensive.
2. **Direct access to a trusted public file** (public-key repository).
   Advantage: no user interaction.
   Problems:
   - The repository must be secure and tamper-proof (otherwise impersonation is still possible),
   - Users must have a secure channel (see Point 1) to initially register their public keys.

Public Key Distribution, II

3. An **on-line trusted server** dispenses public keys on request. The server signs the transmitted keys with its private key.
   Problems:
   - All users must know the server's public verification key
   - The trusted server must be online and may become a bottleneck
   - A communication link must be established with both the server and the intended recipient
   - The server's public-key database may still be subject to tampering
4. **Off-line server and certificates** (certification authorities).
5. Use of **systems implicitly guaranteeing authenticity** of public parameters (ID-based systems).

Option 5 is feasible, but has its own problems. We will focus on Option 4.

Public-Key Infrastructures

Definition 1 (Public-Key Infrastructure (PKI))

A set of techniques and procedures supporting authenticated key management for PKC. Specifically, a PKI supports:

- initialization of system users
- generation, distribution/authentication, and installation of public and private keys
- controlling the use of keys (eg. life cycles of session keys, public and private keys)
- update, revocation, and destruction of keys (eg. managing compromise of private keys)
- storage, backup/recovery, and archival of keys (eg. maintaining an audit trail)

Public-Key Certificates

Definition 2 (Public-Key Certificate)

A data structure consisting of a **data part** (containing at least the user ID and the corresponding public key) and a **signature part** consisting of the digital signature of a *certification authority* over the data part.

A certificate should also include information such as:

- A time-stamp indicating the currency of the certificate (to facilitate key changing and revocation)
- Additional information about the key (key generation algorithm, intended use)
- Key status (for revocation)
- Signature verification information (certification authority's name, signature algorithm used)

Certification Authorities

Definition 3 (Certification Authority (CA))

A trusted third party whose signature on a certificate vouches for the authenticity of the public key bound to the subject entity.

Idea: CA issues public key certificates that may be verified off-line. Users may exchange authentic public keys without having to contact the CA.

Example 1

X.509 is an IETF (Internet Engineering Task Force) standard for certificate-based authentication schemes (used in S/MIME, IPsec, TLS/SSL).

Obtaining Public Keys

Bob uses a public-key certificate to obtain Alice's authentic public key as follows:

1. Acquires the authentic public key of the CA (done only once, eg. pre-loaded in web browsers)
2. Acquires a public key certificate corresponding to Alice over an insecure channel such as a central database, or even directly from Alice
3. Verify the authenticity of Alice's public key:
   (a) Verifies the currency of the certificate using the time-stamp
   (b) Verifies the signature on the certificate using CA's public key
   (c) Verifies that the certificate has not been revoked
4. If all the checks succeed, accepts the public key in the certificate as Alice's public key

Requirements for the Scheme

1. Any participant can read a certificate to determine the name and public key
2. Any participant can verify that the certificate originated from the CA and is not counterfeit
3. Only the CA can create and update certificates
4. Any participant can verify the currency of the certificate

Main Issue / Problem: CA has to be trustworthy!

- not bad for small, closed deployment
- national or international level?

User Registration

Users must register with the CA in a secure manner (typically in person):

- The CA's public key (required for certificate verification) may be obtained at that time
- CA may generate user keys, or certify owner-generated keys (possibly without user revealing the private key)
- May store keys for backup

CA must verify the **binding** between the public and private keys.

CA Hierarchies

Large networks have hierarchies of CAs:

- Tree hierarchy — each node represents a principal whose public key is certified by its parent
- Leaf nodes — end users
- Non-leaf nodes — CAs at various levels and domains (e.g. country level has domains)
  - industry (.com)
  - education (.edu)
  - government (.gov)
  - other organization (.org, .net)
- Two end users can obtain authentic public keys by finding a common ancestor node in the hierarchy

Certificate Revocation

Certificates may need to be revoked before they expire, for the following reasons:

- A user's private key is compromised
- A user is no longer certified by their current CA
- A CA's certificate is assumed to be compromised

Mechanisms for revocation:

- CA maintains a **certificate revocation list** (CRL), available online, signed by the CA
- Alternatively, incremental lists known as **delta revocation lists** are disseminated through the hierarchy
- CA must time-stamp revocations — signatures issued prior to revocation date should be considered valid

Other Key Authentication Mechanisms

Peer authentication of public keys:

- Multiple signatures by different *peers* are attached to public keys
- Used, for example, in PGP via *key rings* with a framework of trust quantification
  - Complicated – trust is hard to quantify and isn't static

Identity based based cryptography

- No need for public key authentication (your ID is your public key)
- Trusted authority generates, holds and knows(!) all private keys

Authentication - Recap

What needs to be authenticated? How is the authentication achieved?

- Messages
  - MACs or hashing with encryption (data integrity)
- Data Origin
  - Digital signatures (also provide non-repudiation)
- Keys
  - Key Distribution Centres (KDCs) for symmetric session keys
  - Public Key Infrastructures (PKIs) or peer authentication for asymmetric keys
  - Trusted authorities for private keys in identity based based crypto
- Entities
  - Authentication protocols (also for access control) ← next

Entity Authentication

We've covered data origin authentication via digital signatures and the frameworks of KDCs or PKIs for key authentication.

We still need protocols for ensuring *entity authentication* within these frameworks.

Authentication Protocols and Nonces

Definition 4 (Authentication protocol)

A sequence of one or more information exchanges used to convince parties of each others' identity.

Authentication may be one-way or mutual. Key issues:

- Confidentiality (e.g. to protect session keys)
- Timeliness (freshness) — to prevent **replay attacks** where a signed message is copied and later resent
  - Ensured via *time stamps* or *nonces*

Definition 5 (Nonce)

A number or bit string that is used only once (usually in a particular message or protocol).

Standardized Authentication Protocols

Notation:

A, B — identities of users A and B, respectively
M — identity of a masquerader (impersonator)
T — identity of a trusted authority
K — authenticated session key produced by the protocol
K_{XY} — key shared by entities X and Y
$E_{K_{XY}}$ — symmetric encryption using key K_{XY}
TS_X — time stamp generated by entity X
N_X — nonce generated by entity X
$cert_X$ — public key certificate of entity X
sig_X — public key signature generated by entity X

"$X \to Y : m$" means that user X sends message m to user Y

Authenticated Session Key Distribution Via KDC

Needham-Schroeder 1978

- Original KDC session key distribution protocol (basis of *Kerberos*[1] session key distribution)
- Utilizes a challenge-response mechanism and symmetric encryption (no public key)
- T plays the role of the KDC

[1] In Ancient Greek mythology, Kerberos is three-headed dog who guards the gates of Hell and prevents dead souls from returning to the world of the living.

Needham-Schroeder Protocol

Protocol:

1. $A \to T : A, B, N_A$
2. $T \to A : E_{K_{AT}}(K, B, N_A, E_{K_{BT}}(K, A))$
3. $A \to B : E_{K_{BT}}(K, A)$
4. $B \to A : E_K(N_B)$
5. $A \to B : E_K(N_B - 1)$

Steps 1,2,3: session key distribution
Steps 4,5: mutual authentication of A and B

Replay Attack on Needham-Schroeder

Suppose M has compromised a previous session key K' and has recorded message 3 from a previous run:

1. $A \to B : E_{K_{BT}}(K', A)$.

Denning, Sacco (1981) — M impersonates A as follows:

1. $M \to B : E_{K_{BT}}(K', A)$ (replay of old, valid message)
2. $B \to M : E_{K'}(N_B)$
3. $M \to B : E_{K'}(N_B - 1)$

Result:

- B accepts K' as a valid session key shared with A
- M can continue to impersonate A successfully.

Denning's & Sacco's Proposed Fix

Uses a time stamp TS_T generated by T instead of A's nonce N_A:

1. $A \to T : A, B$
2. $T \to A : E_{K_{AT}}(K, B, \mathbf{TS_T}, E_{K_{BT}}(K, A, \mathbf{TS_T}))$
3. $A \to B : E_{K_{BT}}(K, A, \mathbf{TS_T})$
4. $B \to A : E_K(N_B)$
5. $A \to B : E_K(N_B - 1)$

Good news: replaying message 3 will no longer work, because B will reject the message if the current time differs greatly from TS_T.

Bad news: a **suppress-replay attack** is possible if B's clock is not tamper-proof. M proceeds as follows:

- Sets B's clock behind and suppress Message 3
- Sends Message 3 when B's clock corresponds to TS_T.

Fix — Combination of Nonces and Expiration Times

Let $time_B$ denote the expiration time for K (determined by B)

1. $A \to B : A, \mathbf{N_A}$
2. $B \to T : B, N_B, E_{K_{BT}}(A, \mathbf{N_A}, \mathbf{time_B})$
3. $T \to A : E_{K_{AT}}(B, \mathbf{N_A}, K, \mathbf{time_B}), E_{K_{BT}}(A, K, \mathbf{time_B}), N_B$
4. $A \to B : E_{K_{BT}}(A, K, \mathbf{time_B}), E_K(N_B)$

Nonces N_A and N_B assure both A and B of session timeliness

Only B needs to check $time_B$, so no clock synchronization needed

In Message 3, the block $E_{K_{BT}}(A, K, time_B)$ serves as a **ticket** that A can use to re-authenticate with B without interaction with T during the time limit specified by $time_B$:

1. $A \to B : E_{K_{BT}}(A, K, time_B), N'_A$
2. $B \to A : N'_B, E_K(N'_A)$
3. $A \to B : E_K(N'_B)$

Kerberos 5

Kerberos is a protocol for authenticated session key distribution via a trusted authority (KDC).

- Utilizes a challenge-response mechanism and *symmetric* encryption
- Simplified version presented here (all non-crypto stuff omitted)
- T plays the role of the KDC; K is the session key with a *validity period* val; both are generated by T

Protocol:

1. $A \to T : A, B, N_A$
2. $T \to A : E_{K_{AT}}(N_A, K, val, B, t)$ where $t = E_{K_{BT}}(K, val, A)$
3. $A \to B : t,\ E_K(A, TS_A)$
4. $B \to A : E_K(TS_A + 1)$

Steps 1, 2, 3: session key distribution.
Steps 3 and 4: mutual key confirmation – both parties encrypt and decrypt with K.

Kerberos (cont'd)

- In message 3, $E_K(A, TS_A)$ serves as an *authenticator* of A to B as only A could have extracted K from $E_{K_{AT}}(K, val, B, t)$

  Similarly, timely decryption of $E_K(TS_A + 1)$ in message 4 provides limited authentication of B to A as only B could have extracted K from the *ticket* $t = E_{K_{BT}}(K, time, A)$.

  However, the IDs and encrypted keys should be properly authenticated with MACs.

- As before, t in message 2 serves as a ticket for A to re-authenticate to B.

Authenticated Diffie-Hellman Key Agreement

Diffie 1992

- Also referred to as *station-to-station* (STS) protocol
- Basis of Internet Key Exchange (IKE) protocol component of IPsec

Public parameters:

- Large prime p, primitive root g of p
- CA's public key (for certificate validation)

Notation:

- A, B – communicating entities (registered with CA)
- $cert_U$ – user U's CA-issued public key certificate (to be validated with CA's public key)
- sig_U – user U's digital signature (to be verified with $cert_U$)

STS Protocol

Protocol (all "(mod p)"s omitted to avoid clutter):

1. $A \to B : g^a$
   - *A selects random integer a, computes* g^a
2. $B \to A : g^b, cert_B, E_K(sig_B(A, B, g^b, g^a))$
   - *B selects random integer b and computes* g^b
   - *B computes shared session key* $K = g^{ab}$ *from* g^a *and b*
   - *B signs both user IDs,* g^b, g^a *with his private key*
   - *B encrypts his signature using the session key K*
3. $A \to B : cert_A, E_K(sig_A(A, B, g^a, g^b))$
   - *A computes shared session key* $K = g^{ab}$ *from* g^b *and a*
   - *A decrypts B's signature using the session key K*
   - *A verifies B's signature using his public key* $cert_B$
   - *A signs both user IDs,* g^a, g^b *with her private key*
   - *A encrypts her signature using the session key K*
   - *B decrypts A's signature using the session key K*
   - *B verifies A's signature using her public key* $cert_A$

Services Provided by STS

Mutual entity authentication (via signed user IDs)

Mutual authenticated key agreement — each party contributes randomness to K, each party signs the key agreement material g^a, g^b

Mutual key confirmation — both parties encrypt and decrypt with K

Perfect forward secrecy — compromise of one session key K or even one of the private keys used for signature generation does not compromise previous session keys as each session key is generated from one-time secrets g^a, g^b.

Note: g^a and g^b also playing the role of nonces to assure freshness

Denial of Service Attack on Original STS

The original version of STS did not include the IDs A and B in the signed messages, thus succumbing to a **denial of service attack** against A whereby an attacker M masquerades as B to A and faces B as himself (Lowe 1994):

1. $A \to M : A, g^a$ **(A thinks M is B)**
   a. $M \to B : M, g^a$ **(M initiates protocol with B as himself)**
2. $B \to M : g^b, cert_B, E_K\left(sig_B(g^b, g^a)\right)$
   a. $M \to A : g^b, cert_B, E_K\left(sig_B(g^b, g^a)\right)$
   (A believes this message is from B due to the signature)
3. $A \to M : cert_A, E_K\left(sig_A(g^a, g^b)\right)$

Result:

- Denial-of-service against A: believes she shares a session key with B.
- B thinks he has participated in an incomplete run with M and is unaware that A is involved at all.

Fix of DoS Attack on STS

This DoS attack is significant if A is a server, as M can cause many false user authentications (and subsequent resource allocations).

Simple fix — include IDs of both participants in the signed messages:

1. $A \to B : g^a$
2. $B \to A : g^b, cert_B, E_K\left(sig_B(\mathbf{A}, \mathbf{B}, g^b, g^a)\right)$
3. $A \to B : cert_A, E_K\left(sig_A(\mathbf{A}, \mathbf{B}, g^a, g^b)\right)$

Previous attack fails: if B and M are included in B's response in message 2, then M cannot use this message to authenticate to A.

General principle (Abadi and Needham):

> *If the identity of a principal is essential to the meaning of a message, it is prudent to mention the principal's name explicitly in the message.*

Lessons Learned

Cryptography in the real world is hard!

- Real-world solutions are often unsatisfactory
- They may be too hard to use, too expensive, too slow, so people won't use them
- Real-world crypto is often poorly implemented and/or poorly used

SSH (Secure Shell)

We will now see a real-world application that puts much of what we've learned together: **SSH** (Secure Shell) is a PKC-based access control system for remote login and file transfer that consists of 3 components:

1. **SSH Transport Layer Protocol** (TLP)
   - algorithm negotiation
   - unilateral authentication (server to client) — client downloads server's public key
   - establishment of shared session key for secure communication
2. **SSH User Authentication Protocol**
   - unilateral authentication (client to server) protected by shared session key
3. **SSH Connection Protocol**
   - interactive applications protected by shared session key

Once the secure channel is set up in step 1, the other two are relatively straightforward.

SSH TLP — Transmission Control Protocol (TCP)

Establishes (as yet insecure) connection between client C and server S:

1. $C \to S : V_C$
2. $S \to C : V_S$
3. $C \to S : I_C$
4. $S \to C : I_S$

Steps 1 & 2: identification

- V_C, V_S: client's and server's SSH protocol and software versions

Steps 3 & 4: algorithm negotiation

- I_C, I_S: lists of algorithms supported for key agreement, encryption, integrity, hashing/compression

For each category, the algorithm chosen is the first one listed in I_C that is also listed in I_S.

SSH TLP — Key Agreement

Unilaterally authenticated Diffie-Hellman, server S to client C:

Protocol (all "mod p"s omitted):

1. $C \to S : g^a$
2. $S \to C : K_S,\ g^b,\ sig_S\big(H(V_C, V_S, I_C, I_S, K_S, g^a, g^b, K)\big)$ where
   - K_S: server's public key
   - $K = g^{ab}$: session key
   - H: hash function
   - V_C, V_S: SSH protocol & software versions
   - I_C, I_S: algorithm lists
3. C verifies authenticity of K_S, validates the server's signature and the hash tag

Note that K_S is not authenticated (beware of man-in-the-middle attacks!)

Management and Validation of Server's Public Keys

Two approaches: PKI or local database

1. Superior solution: public-key certificates
   - Problem: PKI not widely deployed
2. Current solution: each client maintains a local database containing associations between servers and public keys, e.g.
   - `$HOME/.ssh/known_hosts` in Linux
   - `C:\%USERPROFILE%\AppData\Roaming\SSH\HostKeys` in Windows
3. Suggested methods to ensure authenticity of stored public keys:
   - carry authenticated copy on removable storage media (e.g. a USB key or token)
   - obtain public key over an insecure channel, verify over phone (read out hash of obtained public key — unfortunately, this is generally not done)

   Not perfect, but a huge improvement over old applications like `rlogin, rsh, rftp, telnet` etc (which have no or little security!)

Once authenticated Diffie-Hellman is completed, server and client have a shared session key and hence a secure channel.

SSH User Authentication and Connection Protocols

SSH User Authentication Protocol:

- Unilateral authentication (client to server) over the secure channel established by SSH TLP
- Authentication is based on the user proving possession of some cryptographic credential:
  - Public key challenge/response required (private key derived from user's passphrase)
  - Alternative: password based authentication

SSH Connection Protocol:

- standard interactive shell applications over the mutually authenticated secure channel established by the previous two components

Government Recommendations

The Canadian Centre for Cybersecurity (`https://www.cyber.gc.ca/en`) recommends the following algorithms (as of Sept. 2022):

- Encryption: AES in ECB, CFB, OFB, CTR, CBC modes
- Authenticated encryption: CCM, GCM modes
- Key establishment: RSA, DH, MQV, ECC-CDH, ECC-MQV
- Digital signatures: RSA, DSA, ECDSA, hash based in exceptional cases
- Hashing: SHA-2, SHA-3
- Message authentication: HMAC, CMAC, GMAC
- Also recommendations for key derivation, key wrapping, PRBG

See
`https://www.cyber.gc.ca/en/guidance/cryptographic-algorithms-unclassified-protected-protected-b-information-itsp40111`