Computer Security Foundations
Week 11: Public Key Infrastructures and Authentication

Bernardo Portela

L.EIC - 24

Why use a Public Key Infrastructure?

- Public key cryptography presupposes authentic public keys
  - I.e. Alice gets pk_B
  - Means that pk_B is Bob's public key, **and no-one else's**
- Otherwise we get Man-in-the-Middle attacks

Why use a Public Key Infrastructure?

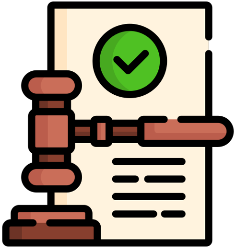
- Public key cryptography presupposes authentic public keys
  - I.e. Alice gets pk_B
  - Means that pk_B is Bob's public key, **and no-one else's**
- Otherwise we get Man-in-the-Middle attacks

- In the real-world, this can be solved in an ad-hoc fashion
  - Manually trust on the public key (e.g. receive this key via a secure, authenticated channel)
  - Rely on systems for public key authentication such as PGP/GPG
  - Or... Public Key Infrastructures

Context for Public-Key Infrastructures

When we need legal coverage $\Rightarrow$ PKI

- Technical norms: which algorithms to use
- Standardization: how technical norms are expected to be applied
- (More) standardization: responsibilities and participant rights
- Laws: formal guarantees and liability on rule violation



It's also a convenient mechanism to know who everyone really is

Public-key Infrastructure - In a Nutshell

- Alice gets a pk_B from *"Bob"*
- She might not trust Bob, but maybe she trusts Charlie
- She can ask Charlie if pk_B comes from Bob
  - If Charlie knows, then he can attest to that
  - Otherwise, just reject pk_B

Charlie can play the role of a **central trusted authority**, and produce signatures that attest to peoples' identities

Public-key Infrastructure - In a Nutshell

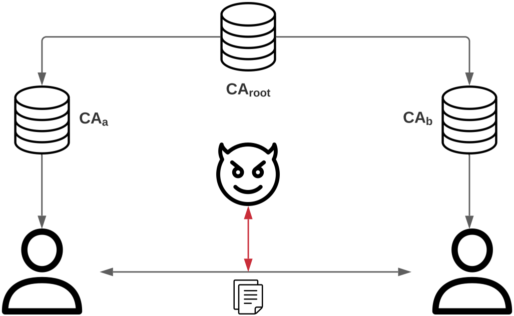
- Alice gets a pk_B from *"Bob"*
- She might not trust Bob, but maybe she trusts Charlie
- She can ask Charlie if pk_B comes from Bob
  - If Charlie knows, then he can attest to that
  - Otherwise, just reject pk_B

Charlie can play the role of a **central trusted authority**, and produce signatures that attest to peoples' identities

Certificates

- Bob sends pk_B and a certificate
- Certificate signed by Charlie, says that pk_B is owned by Bob
- Alice trusts Charlie $\Rightarrow$ Alice trusts pk_B is Bob's

Public-key Infrastructure



- A and B trust CA_{root}
- They might not trust CA_A or CA_B
- Trust hierarchy
  - Root certifies other CAs
  - Sub-CAs certify public keys
  - Alice and Bob exchange certificates

Trusted Computing Base

Bottom-line: We have to assume something!

Trusted Computing Base

Bottom-line: We have to assume something!

Trusted Computing Base (TCB)

- Any security system has it
- Components we will have to *assume* work as expected
- Can have multiple concrete definitions
- Does not mean trust is unwarranted
  - Cryptographic coprocessors
  - Tamper-resistant
  - Standard-compliant APIs
- Trusted hardware not covered, but important to acknowledge!

Public Key Certificates (PKC)

Goal

- Bob sends Alice a public key (pk_B) via insecure channel
- Alice must be assured that Bob holds the secret key (sk_B)

Public Key Certificates (PKC)

Goal

- Bob sends Alice a public key (pk_B) via insecure channel
- Alice must be assured that Bob holds the secret key (sk_B)

Trivial Solution

- Alice maintains an authenticated channel with a **Trusted Third Party (TTP)**
- Bob previously proved TTP he owns pk_B (**Q: How?**)
- Alice can just ask the TTP if pk_B is owned by Bob!

Challenges of Using Certificates

Practical Problems

- Some things solved using algorithms and math
- Some things solved using plain old regulation

Approach: Certificates	1. How to build a channel between Alice and the TTP?
	2. What can we do if the TTP is off-line?
Approach: Regulation	3. How can we be sure that Alice and Bob trust the TTP?
	4. What does it even mean to *"trust"* the TTP?

Solving issues #1 and #2

- PK certificates solve issues #1 and #2 using digital signatures
- Not quite a secure channel, but equally binding conclusions
- Also, service doesn't need to be always on-line!

Solving issues #1 and #2

- PK certificates solve issues #1 and #2 using digital signatures
- Not quite a secure channel, but equally binding conclusions
- Also, service doesn't need to be always on-line!

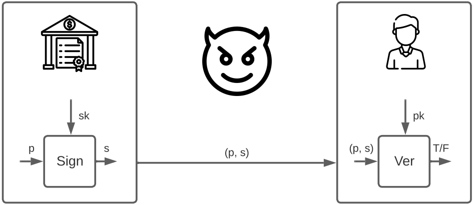
Building the Protocol

- TTP = Certification Authority (CA)
- Bob proves the CA it is in possession of sk_B
  - Usually signing a certificate request containing pk_B
  - The certificate itself is then signed by the CA and given to Bob
  - ... Which can then be verified by Alice.

Concreteness of Certificates

Recall the Importance of Binding

- Signing a message is (pretty much) never enough!
- CA must fix and validate **all** data referred to in the certificate
  - Bob's identity + Bob's public key
  - Specific CA information: CA identity and PKC serial number
  - Validity (start date and validation date)
- CA signs digital document with this information



Anatomy of a Certificate

- Technically a certificate is a document coded with ASN.1
- What is ASN.1?
  - Abstract Syntax Notation 1: platform/language independent
  - Legacy: specification language inherited from protocol norms
  - Norms use ASN.1 to specify data structures (packages)
  - Byte encoding according to Distinguished Encoding Rules

```
TBSCertificate  ::=  SEQUENCE  {
    version         [0]  EXPLICIT Version DEFAULT v1,
    serialNumber         CertificateSerialNumber,
    signature            AlgorithmIdentifier,
    issuer               Name,
    validity             Validity,
    subject              Name,
    subjectPublicKeyInfo SubjectPublicKeyInfo,
    issuerUniqueID  [1]  IMPLICIT UniqueIdentifier OPTIONAL,
                         -- If present, version MUST be v2 or v3
    subjectUniqueID [2]  IMPLICIT UniqueIdentifier OPTIONAL,
                      -- If present, version MUST be v2 or v3
    extensions      [3]  EXPLICIT Extensions OPTIONAL
                      -- If present, version MUST be v3
}
```

Verifying Certificates

Context - Bob sends Alice a certificate with:

- Bob's identity and pk_B
- Validity period (start date and validation)
- Additional metadata
- Signed with a CA trusted by Alice

Verifying Certificates

Context - Bob sends Alice a certificate with:

- Bob's identity and pk_B
- Validity period (start date and validation)
- Additional metadata
- Signed with a CA trusted by Alice

Step-by-step verification by Alice

- Verify if Bob's key and identity matches the certificate
- Verify if the current time is within validity window
- Verify metadata (depends on the application's needs)
- Verify if the CA is trusted
- Obtain pk_{CA} and verify the certificate's signature

Verifying Certificates

Context - Bob sends Alice a certificate with:

- Bob's identity and pk_B
- Validity period (start date and validation)
- Additional metadata
- Signed with a CA trusted by Alice

Step-by-step verification by Alice

- **Purely technological:**
  - Verify if Bob's key and identity matches the certificate
  - Verify if the current time is within validity window
  - Verify metadata (depends on the application's needs)
- **Solved by relying on PKIs:**
  - Verify if the CA is trusted
  - Obtain pk_{CA} and verify the certificate's signature

x.509 Certificates

Details of real-world certificates

- Normalized on x.509 standard
- Transposed to the web by the IETF
- Current version: 3
  - Subject: identity of the user
  - Issuer: identity of the signer CA
  - Validity: period of validity
  - Public key info: the public key
  - Serial: serial number

Manuel Correia
Issued by: TERENA Personal CA 3
Expires: Sunday, 6 March 2022 at 12:00:00 Western European Standard Time
This certificate is valid

> Trust

Details

Subject Name	
Country or Region	PT
Locality	Porto
Organisation	Universidade do Porto
Common Name	Manuel Correia

Issuer Name	
Country or Region	NL
County	Noord-Holland
Locality	Amsterdam
Organisation	TERENA
Common Name	TERENA Personal CA 3

Serial Number	0F 40 8A 05 7F 4E 33 ED 5B 2A 17 AB 4C 29 33 14
Version	3
Signature Algorithm	SHA-256 with RSA Encryption (1.2.840.113549.1.1.11)
Parameters	None
Not Valid Before	Wednesday, 6 March 2019 at 00:00:00 Western European Standard Time
Not Valid After	Sunday, 6 March 2022 at 12:00:00 Western European Standard Time

Public Key Info	
Algorithm	RSA Encryption (1.2.840.113549.1.1.1)
Parameters	None
Public Key	256 bytes: C7 3B 54 0E 3F 07 0D A3 ...
Exponent	65537
Key Size	2 048 bits
Key Usage	Encrypt, Verify, Wrap, Derive
Signature	256 bytes: 02 DA 3A 9B A1 3E 26 30 ...

x.509 Extensions

Additional Certificate Context

- Extensions, which can be set as **critical**
- All extensions have an object identifier (OI)
- If critical, and object is unknown $\Rightarrow$ invalid certificate

x.509 Extensions

Additional Certificate Context

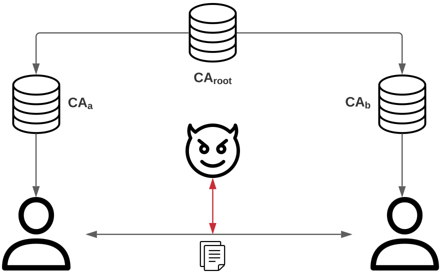
- Extensions, which can be set as **critical**
- All extensions have an object identifier (OI)
- If critical, and object is unknown $\Rightarrow$ invalid certificate

Important Extensions

- Subject/authority key identifier: hash of the public key
- Basic constraints: flag signalling as the original CA certificate
- Key usage: context in which the key is to be used

Definition of PKI

- Allows for a user's identity to be ensured by a certificate
- Produced by a *trusted* certification authority
- Binds a public key to a specific user
- ... and potentially under a given context for usage
- Well-defined responsibilities for all participants



Architecture of a PKI System

```
+---+
| C |                       +------------+
| e | <-------------------->| End entity |
| r |       Operational     +------------+
| t |       transactions          ^
| i |      and management         |   Management
| f |       transactions          |   transactions        PKI
| i |                             |                      users
| c |                             v
| a | =======================  +--+------------+  ==============
| t |                          ^               ^
| e |                          |               |         PKI
|   |                          v               |      management
| & |                       +------+           |       entities
|   | <---------------------|  RA  |<----+     |
| C |  Publish certificate  +------+     |     |
| R |                                    |     |
| L |                                    |     |
|   |                                    v     v
| R |                                +------------+
| e | <------------------------------|     CA     |
| p |   Publish certificate          +------------+
| o |   Publish CRL                     ^      ^
| s |                                   |      |  Management
| i |                +------------+     |      |  transactions
| t | <--------------| CRL Issuer |<----+      |
| o |   Publish CRL  +------------+            v
| r |                                      +------+
| y |                                      |  CA  |
+---+                                      +------+
```

From RFC5280 – Label

- RA - Registration Authority
- CA - Certification Authority
- CRL - Certificate Revocation List

Operational Transactions/Management

- How are certificates distributed and stored?
- Multiple protocols, with concrete specifications
  - In public repositories (e.g. LDAP) or simply included by applications/OSs
  - Can be transferred by applications in specific protocols (HTTPS, FTP, MIME)
  - Must be encoded in a way that ensures interoperability

Operational Transactions/Management

- How are certificates distributed and stored?
- Multiple protocols, with concrete specifications
  - In public repositories (e.g. LDAP) or simply included by applications/OSs
  - Can be transferred by applications in specific protocols (HTTPS, FTP, MIME)
  - Must be encoded in a way that ensures interoperability
- Actually, we have already covered some examples!
  - In TLS, its RFC specifies how certificates can be exchanged
    - RFC (Request For Comments) is a public document denoting how protocols should be implemented
  - In S/MIME, certificates are included in PKCS#7 attachments
  - OSs/browsers manage certificates in secure components
    - It's why it's important to have a **legitimate** operative system!

Meeting the Requirements

Can we match the requirements of this PKI?

- Users must establish some communication with the CA
- Alice must be able to check a CA signature within a certificate

Meeting the Requirements

Can we match the requirements of this PKI?

- Users must establish some communication with the CA
- Alice must be able to check a CA signature within a certificate

Solution

- All public keys are encoded in X.509 certificates
- Some (special) certificates contain public keys of CAs
- Alice gets pk_{CA} from another certificate
- She can then use pk_{CA} to verify the signature in Bob's certificate, and thus bind it to Bob
- The verification succeeds $\Rightarrow$ Alice can use pk_B

Initializing PKIs

How can we “trust” this CA?

- Alice gets the certificate via secure channel
- Alice trusts the CA certificate implicitly

Many examples

- OSs come with quite a lot of pre-installed CA certificates
- CC contains authority certificates managed by the state

These are some of many examples of how to initialize trust on CAs

Certificate Chains

Real instances not as simple as our example

- Intialization of Alice considers *root* certificates
  - *root* certificates are the baseline for trust
  - Alice trusts them implicitly
  - They are self-signed

Certificate Chains

Real instances not as simple as our example

- Intialization of Alice considers *root* certificates
  - *root* certificates are the baseline for trust
  - Alice trusts them implicitly
  - They are self-signed

Underlying Assumptions

- **Anyone can generate a self-signed certificate**
  - This week's tasks will include this!
- Validating a root/self-signed certificate implies:
  - Trusting that the associated *sk* belongs to the CA
  - Trusting that this CA is trustworthy ⇒ and thus its produced/signed certificates will also be trustworthy

Multi-level Certificate Chains

Usually, Root CAs do not authenticate end-users directly

CA Hierarchy

If CA_A signs CA_B, trust in $B \leq$ trust in A

Multi-level Certificate Chains

Usually, Root CAs do not authenticate end-users directly

CA Hierarchy

If CA_A signs CA_B, trust in $B \leq$ trust in A

Real-world complexities

- Root CAs manage sub-CAs that will validate users in different contexts
- We can have multiple hierarchical levels:
  - To authenticate Bob's key, Alice gets Bob's certificate
  - Bob's certificate is signed by some CA_A
  - If Alice trusts CA_A, it's all good. Otherwise...

Multi-level Certificate Chains

Usually, Root CAs do not authenticate end-users directly

CA Hierarchy

If CA_A signs CA_B, trust in $B \leq$ trust in A

Real-world complexities

- Root CAs manage sub-CAs that will validate users in different contexts
- We can have multiple hierarchical levels:
  - To authenticate Bob's key, Alice gets Bob's certificate
  - Bob's certificate is signed by some CA_A
  - If Alice trusts CA_A, it's all good. Otherwise...
  - To authenticate CA_A's key, Alice gets CA_A's certificate
  - CA_A's certificate is signed by some CA_B
  - If Alice trusts CA_B, it's all good. Otherwise...

Multi-level Certificate Chains

Usually, Root CAs do not authenticate end-users directly

CA Hierarchy

If CA_A signs CA_B, trust in $B \leq$ trust in A

Real-world complexities

- Root CAs manage sub-CAs that will validate users in different contexts
- We can have multiple hierarchical levels:
  - To authenticate Bob's key, Alice gets Bob's certificate
  - Bob's certificate is signed by some CA_A
  - If Alice trusts CA_A, it's all good. Otherwise...
  - To authenticate CA_A's key, Alice gets CA_A's certificate
  - CA_A's certificate is signed by some CA_B
  - If Alice trusts CA_B, it's all good. Otherwise...
  - To authenticate CA_B's key, Alice gets CA_B's certificate
  - CA_B's certificate is signed by some CA_C
  - If Alice trusts CA_C, it's all good. Otherwise...

Certificate Revocation

- Certificates are **always** invalid after their validity period
- How can we actively invalidate a certificate?
  - Secret keys are lost
  - CAs become compromised
  - Metadata stops being valid
- We need to ensure **certificate revocation**

Certificate Revocation Lists

Done via Certificate Revocation Lists (CRLs)

- CA periodically publishes certificate black-list
- How to know where the most recent CRL is?
- What can we do if we don't have access to CRLs?

Certificate Revocation Lists

Done via Certificate Revocation Lists (CRLs)

- CA periodically publishes certificate black-list
- How to know where the most recent CRL is?
- What can we do if we don't have access to CRLs?

Three real-world solutions

- Trusted Service Provider Lists (TSL)
  - Frequently updated certificate white-list
  - Used in small/closed communities (e.g. banking)
- On-line Certificate Status Protocol (OCSP)
  - Secure server checks revocation
  - Used in organizational contexts (eGov)
- Certificate pinning:
  - Individually managed by web servers/browsers/apps
  - Identify critical certificate revocation (e.g. Google)

Alternative: Pretty Good Privacy

Pretty Good Privacy

- Privacy and authentication in data communication
- Decentralized *web of trust*
- PGP fingerprint
  - A short public key
  - Printed on business cards

Look at that subtle off-white coloring. The tasteful thickness of it

Web of Trust

- Direct trust established by secure channels
- Indirect trust by vetting by direct trustees
- No single point of failure, but hard to use in practice
- Some applications, but not too widespread

Key Takeaways

- Authentic public keys $\rightarrow$ PKIs
  - Legal coverage
  - Specifies algorithms
  - Standardizes usability

Key Takeaways

- Authentic public keys $\rightarrow$ PKIs
    - Legal coverage
    - Specifies algorithms
    - Standardizes usability
- Certificates ensure authenticity
    - X.509 Certificates
    - Characterise all user identification and usage conditions

Key Takeaways

- Authentic public keys $\rightarrow$ PKIs
  - Legal coverage
  - Specifies algorithms
  - Standardizes usability
- Certificates ensure authenticity
  - X.509 Certificates
  - Characterise all user identification and usage conditions
- Public-Key Infrastructure
  - Holds everything together
  - (Root) CAs trusted by assumption
  - Chain of validations until the end-user
  - Root CAs pre-installed (e.g. by default in browsers)

Key Takeaways

- Authentic public keys → PKIs
  - Legal coverage
  - Specifies algorithms
  - Standardizes usability
- Certificates ensure authenticity
  - X.509 Certificates
  - Characterise all user identification and usage conditions
- Public-Key Infrastructure
  - Holds everything together
  - (Root) CAs trusted by assumption
  - Chain of validations until the end-user
  - Root CAs pre-installed (e.g. by default in browsers)
- Certificate Revocation Lists (CRLs)
  - Periodically published and checked
  - Used to immediately invalidate certificates
  - Challenges in using them correctly

Access Control

Authentication

Determining whether a user should be allowed access to a system

- Local machines
  - Something you know / have / are
- **Through network - security protocols**

Access Control

Authentication

Determining whether a user should be allowed access to a system

- Local machines
  - Something you know / have / are
- **Through network - security protocols**

Authorization

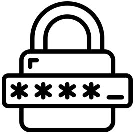
Deciding what actions a user can perform in the system

- Fine-grained set of restrictions to system resources
  - Check system characteristics
  - Read/write data
  - Alter configurations

Authentication Methods

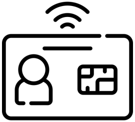
Something you know

- Passwords (single or multi-word)
- Security question



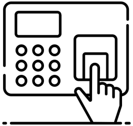
Something you have

- Smart Cards
- Crypto Tokens



Something you are

- Fingerprint
- Other biological data



Protocols

- Human protocols - Rules followed in human interactions
  - *E.g.* Asking a question in class

Protocols

- Human protocols - Rules followed in human interactions
  - *E.g.* Asking a question in class
- Networking protocols - Rules followed in networked communication systems
  - *E.g.* HTTP, FTP, etc.

Protocols

- Human protocols - Rules followed in human interactions
  - *E.g.* Asking a question in class
- Networking protocols - Rules followed in networked communication systems
  - *E.g.* HTTP, FTP, etc.
- Security Protocol - the (communication) rules followed in a security application
  - *E.g.* SSL, IPSec, SSH, Kerberos, etc.

Authentication Protocols

ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Is the pin correct?

Authentication Protocols

ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Is the pin correct?
    - **YES** - Perform transaction
    - **NO** - Machine eats your card (eventually)

Authentication Protocols

ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Is the pin correct?
    - **YES** - Perform transaction
    - **NO** - Machine eats your card (eventually)

Enter the NSA

1. Insert badge into reader
2. Enter PIN
3. Is the pin correct?

Authentication Protocols

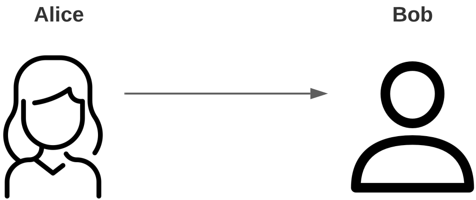
ATM Machine Protocol

1. Insert ATM card
2. Enter PIN
3. Is the pin correct?
    - **YES** - Perform transaction
    - **NO** - Machine eats your card (eventually)

Enter the NSA

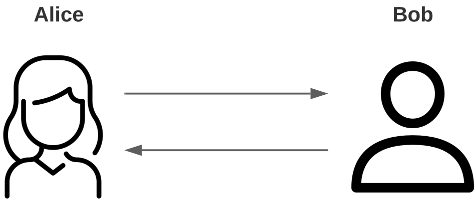
1. Insert badge into reader
2. Enter PIN
3. Is the pin correct?
    - **YES** - Open the door
    - **NO** - Get immediately shot by guard

Authentication Protocols - For real now



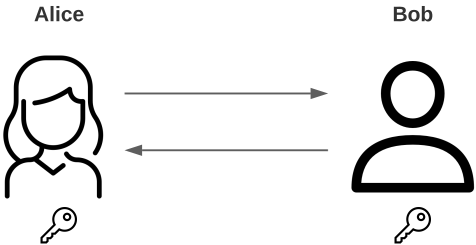
- Alice must prove her identity to Bob
  - They can be humans or computers

Authentication Protocols - For real now



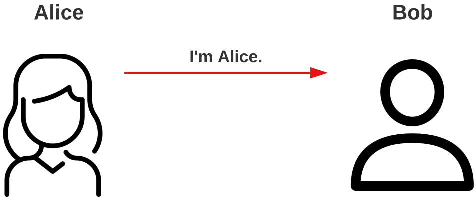
- Alice must prove her identity to Bob
  - They can be humans or computers
- May also require Bob to prove its identity to Alice
  - A.k.a. mutual authentication

Authentication Protocols - For real now



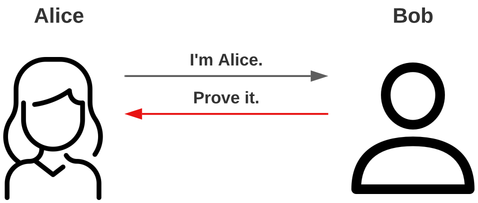
- Alice must prove her identity to Bob
  - They can be humans or computers
- May also require Bob to prove its identity to Alice
  - A.k.a. mutual authentication
- Often entails establishing a session key
  - For cryptographic purposes

A Naive Authentication Protocol



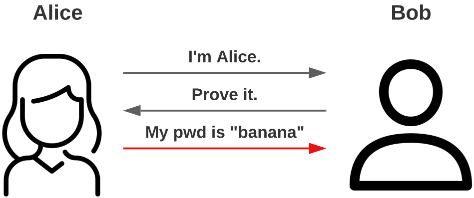
- Simple and (perhaps) OK for a stand-alone machine

A Naive Authentication Protocol



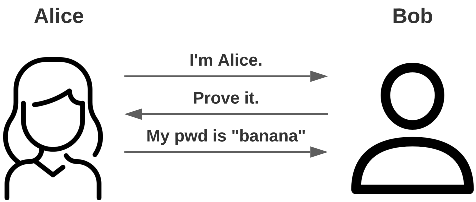
- Simple and (perhaps) OK for a stand-alone machine

A Naive Authentication Protocol



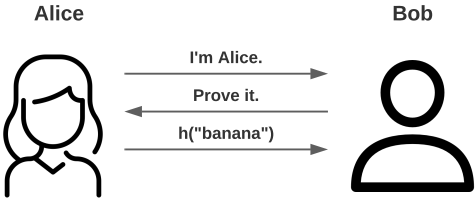
- Simple and (perhaps) OK for a stand-alone machine

A Naive Authentication Protocol



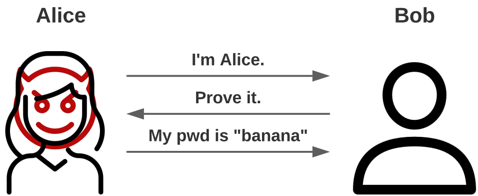
- Simple and (perhaps) OK for a stand-alone machine
- Highly insecure for a networked system
  - Subject to replay attacks
  - Bob must know Alice's password (explicitly)

A (still pretty) Naive Authentication



- This approach hides Alice's password
    - From both Bob, and the adversary!
- But it's subject to replay attacks...

Replay Attacks



- This is an example of a **replay** attack
  - The adversary observed the interaction
  - Used the messages to repeat a communication pattern
- How can we prevent replay attacks?

Challenge-Response

To prevent replay, we leverage a technique called **challenge-response**

- Suppose Bob wants to authenticate Alice (our setting)
- Bob sends a *challenge* to Alice
- Alice must respond to the *challenge* according to its password

Challenge-Response

To prevent replay, we leverage a technique called **challenge-response**

- Suppose Bob wants to authenticate Alice (our setting)
- Bob sends a *challenge* to Alice
- Alice must respond to the *challenge* according to its password

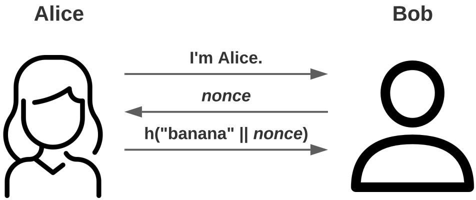
Challenge

Challenge is chosen such that...

- Replay is not possible
- Only Alice can provide the correct response
- Bob can (efficiently) verify the response

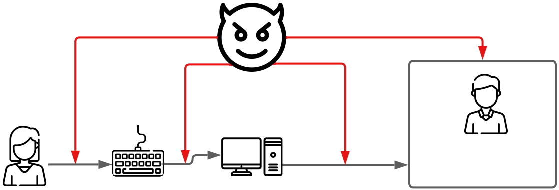
Using Nonces

Nonce: A **n**umber that is only used **once**.



- Nonce is the challenge
  - Every request for authentication must use a different nonce
- The hash is the response
  - The message used for the first authentication will not work for any of the following ones
  - Collision-resistant hashes!
- Bob must know Alice's pwd to verify.

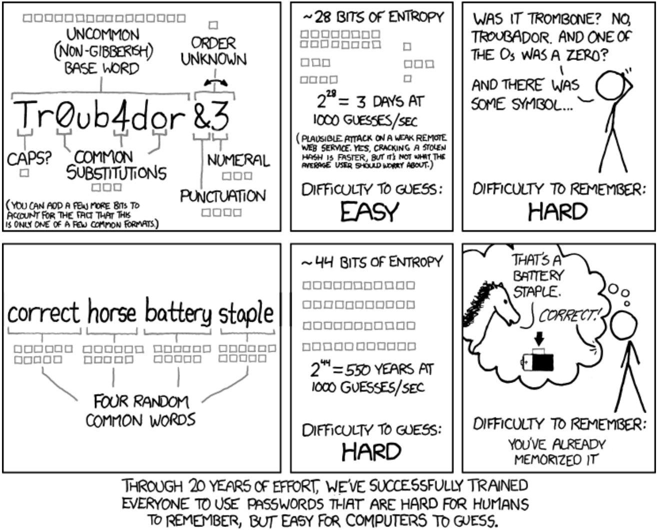
Password Attacks



Attacks come in many flavours

- In-person
- Keyloggers
- Network packet sniffing
- Server (Bob) hacking

Strong Passwords



Password Guessing

Keyloggers

- HW keyloggers - devices between keyboard and computer
- SW keyloggers - malware intercepting keystrokes
- Check stored passwords, browser cache, etc.

Password Guessing

Keyloggers

- HW keyloggers - devices between keyboard and computer
- SW keyloggers - malware intercepting keystrokes
- Check stored passwords, browser cache, etc.

Dictionary attacks

- Server stores hashes of passwords $H(pw)$
- Server gets breached! How many passwords to test?
  - For digit and letters: $26 + 26 + 10 = 64$
  - 64^n for passwords of length n. For $n = 6, 2^{36}$ possibilities
  - But you can pre-compute them.
  - I.e. accumulate hashes of common (and uncommon) passwords throughout the years

Password Guessing

Keyloggers

- HW keyloggers - devices between keyboard and computer
- SW keyloggers - malware intercepting keystrokes
- Check stored passwords, browser cache, etc.

Dictionary attacks

- Server stores hashes of passwords $H(pw)$
- Server gets breached! How many passwords to test?
  - For digit and letters: $26 + 26 + 10 = 64$
  - 64^n for passwords of length n. For $n = 6, 2^{36}$ possibilities
  - But you can pre-compute them.
  - I.e. accumulate hashes of common (and uncommon) passwords throughout the years
- **Huge datasets** allow for instantly testing passwords

Countermeasure: Salting passwords

- Instead of just blindly storing $H(pw)$
- Generate a random (relatively short) r
- Store $(r, H(r||pw))$
- **How can this help defend against dictionary attacks?**

Countermeasure: Salting passwords

- Instead of just blindly storing $H(pw)$
- Generate a random (relatively short) r
- Store $(r, H(r||pw))$
- **How can this help defend against dictionary attacks?**
- No longer realistic to pre-compute
  - requires the consideration of all salts for common passwords
  - for N common passwords, assuming a salt of one byte: $N * 2^8$
  - for a 4-byte salt: $N * (2^8)^4$

Note: after breaching a server, the attacker does not have to compute all possible salts, as each salt becomes known. This is <u>only</u> to prevent <u>pre-computation</u>

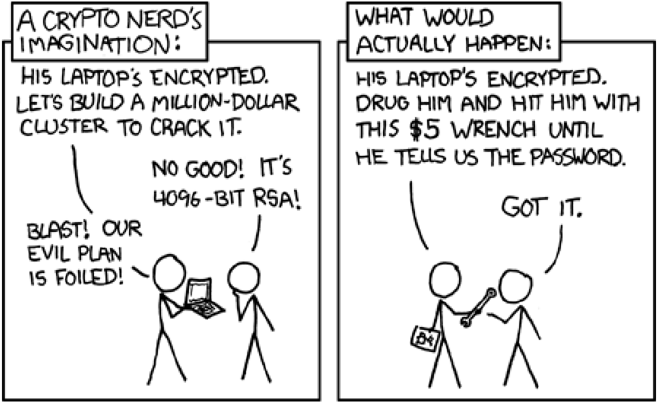
Phishing

Adversary convinces Alice to just give him the key

URL tampering

- HTTP
    - Present an incorrect server, without the certificate
    - Much harder to do with modern practices
- HTTPS
    - Change website to a "similar looking one"
    - sigarra.up to sigarra.vp
    - Homoglyphs - register similar domains to largely popular ones

Pragmatism


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

Computer Security Foundations
Week 11: Public Key Infrastructures and Authentication

Bernardo Portela

L.EIC - 24