1 Overview

In the last lecture we discussed secure hash functions, collision resistance, the birthday problem, message authentication codes and digital signatures.

In this lecture we discuss a vulnerability in RSA digital signatures, a fix for the vulnerability, and how to verify the authenticity of public keys.

2 An RSA Vulnerability

RSA Digital Signatures:

\[ \text{Sign}(m): \quad c \leftarrow m^d \mod n \]
\[ \text{Verify}(m, c): \quad \text{True if } m = c^e \mod n, \text{false otherwise} \]

2.1 Digital signature forgery

It is easy to forge a message-signature pair \((m, c)\) that verifies. Example: \((c^e \mod n, c)\)

However, since the plaintext message often has format, this approach doesn’t work well.

Suppose we are given a signing oracle, something that will sign any message of our choosing.

**Question:** How can we construct the signature for a message that we didn’t ask the signing oracle to sign?

We want \(m^d \mod n\).

Ask the signing oracle to sign \(m_1\). \(\text{Sign}(m_1) = m_1^d \mod n\).

Find \(x\) such that \(x \times m_1 = m \mod n\).

Ask the signing oracle to sign \(x\). \(\text{Sign}(x) = x^d \mod n\).

\[
\begin{align*}
\text{Sign}(m) &= \text{Sign}(x) \times \text{Sign}(m_1) \\
&= (x^d \mod n) \times (m_1^d \mod n) \\
&= (x^d \times m_1^d) \mod n \\
&= (x \times m_1)^d \mod n \\
&= (m)^d \mod n
\end{align*}
\]
2.2 Fixing This Vulnerability

We want to destroy the ability to construct RSA signatures due to the properties of multiplication.
We want small differences in messages to map to wildly different signatures.
Secure hash functions fulfill this desire.

\[
\text{Sign}(m): c \leftarrow (h(m))^d \mod n.
\]
\[
\text{Verify}(m, c): \text{True if } h(m) = c^e \mod n. \text{ False otherwise.}
\]

3 Certificates

Problem: What happens if a published public key attributed to Alice is not Alice's actual public key?

The common solution to this question is to have a trusted third party verify the authenticity of the public key. Browsers come preloaded with the public keys of certain trusted companies so that the browser can verify the authenticity of certificates. Verisign is a certificate authority that administers certificates. Certificates contain the public key of Alice, the ID of Alice, and the signature of those items from a certificate authority such as Verisign.

\[
\text{Certificate}_A = \{ [k_A, ID_A]^{-1}_{\text{Verisign}}, k_A, ID_A \}
\]

Typically, the ID is the domain name. Certificates contain other information as well.

3.1 Using Certificates

There are three initial steps in the https protocol:

1. Download Certificate.
2. Compute Verify(m, c)
3. Match domain name to the certificate’s domain name.

3.2 Certificate Errors

There are four types of certificate errors, ordered from most malicious to least malicious:

1. Verify(m, c) returns false
2. Domain name mismatch
3. Self-signed certificate: The certificate was signed by a party whose key cannot be accessed by the browser.
4. Expired certificate
3.3 Certificate Chains

Not every certificate needs to be signed by Verisign or another certificate authority. It's possible for certain verified users to issue certificates themselves.

Example: Faculty at UC Davis who wish to have their certificates signed can have them signed by UC Davis, which had its own certificate signed by Verisign. When a browser attempts to access that faculty's webpage, it sees a certificate signed by UC Davis. The browser gets UC Davis's certificate, which has been signed by Verisign, and verifies it.