Introduction to Cryptography

Alvin Lin

January 2018 - May 2018

Digital Signatures

Suppose Alice orders a coffee maker from Bob's website. After seeing the coffee maker, Alice states that she has never ordered it. How can Bob prove towards a judge that Alice has never ordered a pink car? Symmetric cryptography fails because both Alice and Bob can be malicious. This can only be achieved with public key cryptography. The straightforward use of public key encryption provides **confidentiality** but not **authentication**, however public key cryptography can be used to provide authentication.

Authentication

For a given message X, a digital signature is appended to the message just like a conventional signature. Only the person with the private key should be able to generate the signature. The signature must change for every document and is generated as a function with the message X and the private key as input. The public key and the message X are the inputs to the verification function. This provides authentication since only the holder of the private key would have been able to provide that signature.

To provide both confidentiality and authentication, Alice can encrypt the message M first using its private key which provides a digital signature, and then using Bob's public key, which provides confidentiality. The disadvantage is that the public key algorithm must be exercised four times during each communication.

Goals of Digital Signatures

1. Confidentiality: Information is kept secret from unauthorized parties.

2. Integrity: Ensures that a message has not been modified in transit.

3. Message Authentication: Ensures that the sender of a message is authentic, also known as data origin authentication.

4. Non-repudiation: Ensures that the sender of a message cannot deny the creation of the message.

Additional goals of a secure system:

1. Identification/entity authentication: Establishing and verifying the identity of an entity, whether it be a person, a computer, or a credit card.

2. Access control: Restricting access to resources only to privileged entities.

3. Availability: Providing readily available access to the system.

4. Auditing: Capability of providing evidence about security relevant activities by keeping logs about certain events.

5. Physical Security: Providing security against physical tampering.

6. Anonymity: Providing protection against discovery and misuse of identity.

Naive RSA Digital Signature Scheme

To generate the private and public key, the same key generation scheme as RSA is used. The signature is generated by encrypting the message X with the private key K_{pr}:

$$S = signature_{kpr}(X) = X^d \mod n$$

The signature S is appended to the message X. To verify the signature, decrypt it with the public key K_{pub}:

$$X' = verification_{kpub}(S) = S^e \mod n$$

If the message X matches the decrypted signature X', then the signature is valid. The security of the signature has the same constraints as RSA encryption. n needs to be at least 1024 bits long and the signature S needs to be at least 1024 bits long. Signature verification can be done very quickly if a small number is chosen for the public key.

While effective, RSA digital signatures still have the problem of message size and existential forgery. They only work if the message X is a number less than n. Because of the homomorphic property of RSA, if a hacker sees two message signatures, they can compute the signature on the message without knowing the private key. Additionally, suppose a hacker picks some signature S and computes $X = S^e \mod n$ using Bob's public key $K_B = e$. They can send the message X and signature S to impersonate Bob. A person trying to verify the signature will compute $X' = S^e \mod n$ and find that $X' = X$. While the message may not make sense, the signature will match even though it did not come from Bob. Attackers can generate valid message-signature pairs, but are only able to choose signatures and not the message encoded. These problems can be combatted by using hash functions or padding schemes to prevent attackers from generating valid key-signature pairs.

Public Key Algorithms

Algorithm	Encryption/Decryption	Digital Signature	Key Exchange
RSA	Yes	Yes	Yes
Elliptic Curve	Yes	Yes	Yes
Diffie-Hellman	No	No	Yes
DSS	No	Yes	No

El Gamal Digital Signatures

The security using El Gamal for digital signatures is based on the hardness of the discrete logarithm problem. If Alice wants to sign a message, she needs to generate a key:

1. Alice chooses a large prime p
2. Alice chooses a generator g for a large-order subgroup $\mathbb{Z}_p^*$
3. Alice chooses a secret random number a
4. Alice computes $y = g^a \mod p$
5. Alice's public key is (p, g, y) and her private key is a

She can then sign a message m using her private key:

1. Alice chooses a secret random number k with $gcd(k, p-1) = 1$

2. Alice choose $R = g^k \mod p$
3. Alice computes k^{-1} where $k \cdot k^{-1} \equiv 1 \mod (p-1)$
4. Alice computes $S = k^{-1}(Hash(X) - aR) \mod (p-1)$
5. Alice's signature for X is (R, S)

Bob can then verify the signature (R, S) by using y from Alice's public key to check if:

$$g^{Hash(X)} = y^R R^S \mod p$$

Digital Signature Algorithm (DSA)

The digital signature algorithm is based on the El Gamal signature scheme. It was proposed by the National Institute of Standards and Technology and uses a 320 bit signature. Signature verification for this algorithm is significantly slower compared to RSA. To generate a key:

1. Generate a prime p with $2^{1023} < p < 2^{1024}$.
2. Find a prime divisor q of $p-1$ with $2^{159} < q < 2^{160}$.
3. Find an integer α with $order(\alpha) = q$.
4. Choose a random integer d with $1 < d < q$.
5. Compute $\beta = a^d \mod p$.
6. $k_{pub} = (p, q, \alpha, \beta) \quad k_{pr} = d$

Given a message x, a private key d and a the public key (p, q, α, β), generating a signature requires the following:

1. Choose an integer as a random ephemeral key k_E with $0 < k_E < q$.
2. Compute $r \equiv (\alpha^{k_E} \mod p) \mod q$.
3. Compute the signature $s \equiv (SHA(x) + d \cdot r)(k_E)^{-1} \mod q$.

The message signature consists of the pair (r, s) $SHA(x)$ denotes the hash function SHA-1 which computes a 160-bit fingerprint of message x. To verify the signature:

1. Compute the auxiliary value $w \equiv s^{-1} \mod q$.

2. Compute the auxiliary value $u_1 \equiv w \cdot SHA(x) \mod q$.
3. Compute the auxiliary value $u_2 \equiv w \cdot r \mod q$.
4. Compute $v \equiv (a^{u_1} \cdot \beta^{u_2} \mod p) \mod q$.
5. If $v \equiv r \mod q$, the signature is valid. Otherwise, it is invalid.

Elliptic Curve Digital Signature Algorithm (ECDSA)

The elliptic curve digital signature algorithm is based on elliptic curve cryptography. Bit lengths in the range of 160-256 bits can be chosen to provide security equivalent to 1024-3072 bit RSA. One signature consists of two points, hence the signature is twice the used bit length. The shorter bit length of this algorithm often results in shorter processing time. To generate a key using this algorithm:

1. We use an elliptic curve E with a modulus p, coefficients a and b, and a point A which generates a cyclic group of prime order q.
2. Choose a random integer d with $1 < d < q$.
3. Compute $B = dA$.
4. $k_{pub} = (p, a, b, q, A, B) \quad k_{pr} = d$

Generating a message signature is very similar to the DSA algorithm:

1. Choose an integer as a random ephemeral key k_E with $1 < k_E < q$.
2. Compute $R = k_E A$
3. Let $r = x_R$ from the message x.
4. Compute the signature $s \equiv (SHA(x) + d \cdot r)(k_E)^{-1} \mod q$.

To verify the signature:

1. Compute the auxiliary value $w \equiv s^{-1} \mod q$.
2. Compute the auxiliary value $u_1 \equiv w \cdot SHA(x) \mod q$.
3. Compute the auxiliary value $u_2 \equiv w \cdot r \mod q$.
4. Compute $P = u_1 A + u_2 B$.
5. If $x_P \equiv r \mod q$, the signature is valid. Otherwise, it is invalid.

Digital Signature Standard (DSS)

The federal government standards for digital signatures specifies a suite of algorithms that can be used to generate a digital signature. The DSS permits the usage of the three following digital signature algorithms:

1. RSA
2. Digital Signature Algorithm (DSA)
3. Elliptic Curve Digital Signature Algorithm (ECDSA)

For hash functions, the Digital Signature Standard permits the usage of SHA-1 or SHA-2 hash functions. With RSA, the standard mandates a 1024, 2048, or 3072-bit modulus n since the security is based on the hardness of integer factorization. DSA uses a subgroup $\mathbb{Z}_p^*$ with order q and is based on the hardness of the discrete logarithm problem in $\mathbb{Z}_p^*$, which the standard allows to be either any of the following:

- $p = 1024, q = 160$
- $p = 2048, q = 224$
- $p = 2048, q = 256$
- $p = 3072, q = 256$

For ECDSA, the standard recommends the curves P-192, P-224, P-256, P-384, P-521 with bit sizes q. The security of ECDSA is based on the hardness of the discrete logarithm problem in the elliptic curve group. It is worthwhile to note that these systems allow messages to be authenticated, but not the public keys. An attacker can still execute a man-in-the-middle attack if they can intercept the public keys being exchanged.

You can find all my notes at `http://omgimanerd.tech/notes`. If you have any questions, comments, or concerns, please contact me at alvin@omgimanerd.tech