Justin Ciocoi

Dec. 3, 2023

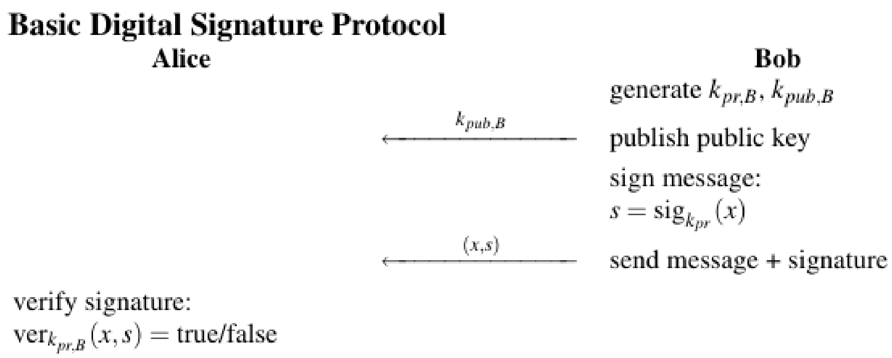
CSCI 360 Textbook Notes

Chapter 10: Digital Signatures

10.1: Introduction

- This section will describe why digital signatures are needed, why they must be based on asymmetric cryptography, and various fundamental principles of digital signature schemes
- **10.1.1: Why Symmetric Cryptography is not Sufficient**
    - Of the crypto schemes that have thus far been covered, all have had one of two goals:
        - To encrypt some data
        - To establish a shared key between two parties
    - While this provides some services, there are other security needs which we will deem security services
    - Under modern symmetric cryptography schemes, we have a level of security protecting communicating parties from outside attacks
    - However, symmetric schemes do not provide either party with any protection against *each other*
    - In order to prove to a neutral third party that one of the two parties sent a specific message, symmetric cryptography is not sufficient, since both parties share the key k_{ab} and can thus both generate messages encrypted in the same manner with the same key
- **10.1.2: Principles of Digital Signatures**
    - The process of digital signatures starts with one party signing a message, x
    - The signature algorithm is a function of this party's private key, k_{pr}, so assuming this key is kept private, this party is the only one who can sign message x on their behalf

- x and k_{pr} are inputs into the signature function, and after signature, the signature s is appended to the message and the pair (x, s) is sent
- The signature is only of use to the receiving party if they have a method by which they can *verify* the signature's validity
- Thus, the receiving party has a verification function which will accept x, s, and, k_{pub} as inputs
  - If x was actually signed with the private key which belongs to the public verification key, then the function returns `true`
  - Otherwise, it returns `false`
- Here, we can see a basic visualization of the protocols outlined above:



- A signed message can unambiguously be traced back to its originator since a valid signature can only be computed with the unique signer's private key
- Each of the three popular public-key algorithm families (integer factorization, discrete logarithms, and elliptic curves) allows us to construct digital signature schemes

- **10.1.3: Security Services**
  - Let us first discuss the various security services that may be provided by cryptographic schemes in general
    - **Confidentiality:** Information is kept secret from all but authorized parties
    - **Integrity:** Messages have not been modified in transit
    - **Message Authentication:** The sender of a message is authentic. An alternative term is *data origin authentication*

- **Nonrepudiation:** The sender of a message can not deny the creation of the message
- **Identification/Entity Authentication:** Establish and verify the identity of an entity, e.g., a person, a computer or a credit card
- **Access Control:** Restrict access to the resources to privileged entities
- **Availability:** Assures that the electronic system is reliably available
- **Auditing:** Provide evidence about security-relevant activities, e.g., by keeping logs about certain events
- **Physical Security:** Provide protection against physical tampering and/or responses to physical tampering attempts
- **Anonymity:** Provide protection against discovery and misuse of identity

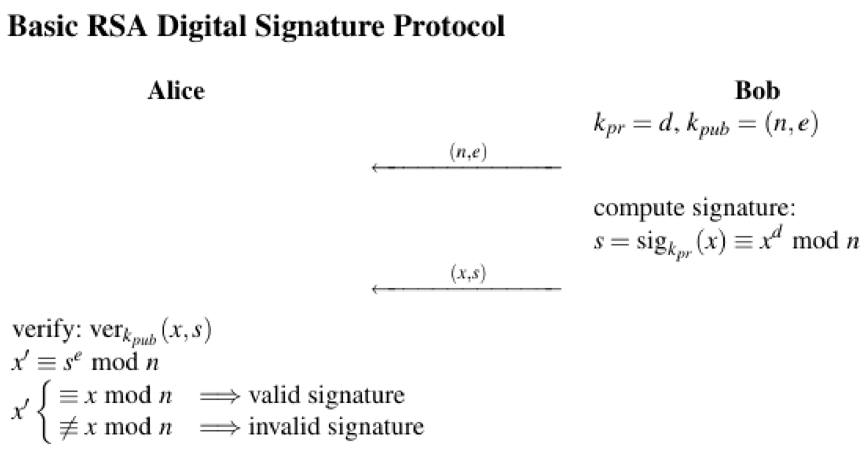
- Which services are desired in a given system is heavily application specific and can vary greatly in different environments or implementations

10.2: The RSA Signature Scheme

- **10.2.1: Schoolbook RSA Digital Signature**
  - First, let us imagine a sending party wants to send a signed message and after set up has the following parameters:

$$\text{Private Key: } k_{pr} = (d)$$
$$\text{Public Key: } k_{pub} = (n, e)$$

  - The basic protocol will be achieved as shown in the following diagram



- As we can see, the sender computes their signature, s by RSA-encrypting x with his private key k_{pr}
- The receiver then calculates x', and if $x = x'$, confirms the validity of the signature
- Let us look at the verification operations:

$$s^e = (x^d)^e = x^{de} \equiv x \bmod n$$

which we can confirm the validity of due to the mathematical relationship between the public and private key, namely:

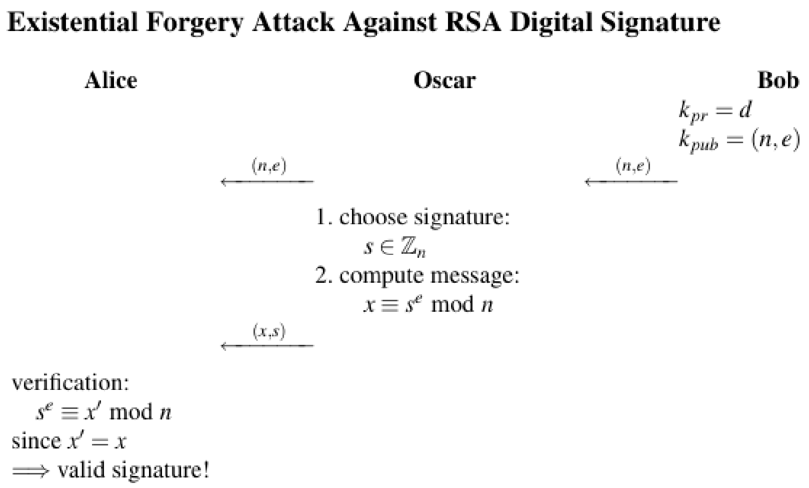
$$d \cdot e \equiv 1 \bmod \phi(n)$$

which means that raising any integer, x to the $(de)^{th}$ power yields itself again, showing that

$$x^{de} \equiv x \bmod n$$

- **10.2.2: Computational Aspects**
  - The signature in RSA digital signature schemes is as long as the modulus n, or roughly $log_2(n)$ bits
  - Using RSA with short public keys is particularly useful, since while a message may only be signed once, verification could happen many times, and thus the short public key can greatly speed up the verification function
- **10.2.3: Security**
  - *Algorithmic Attacks*
    - This group of attacks attempts to break RSA by computing the private key, d
    - The most general of these attacks is an attacker factoring the modulus n into primes p and q, after which the private key d can be computed using the public key e
    - Thus, the modulus n must be sufficiently large to prevent factoring attacks
  - *Existential Forgery*
    - This attack allows an attacker to generate a valid signature for a *random* message x

        - Here we can see a basic visualization of these types of attacks



        - In this scheme, the attacker chooses the signature and then computes the message, so while the semantics of the message cannot be controlled, this is still a clearly undesirable quality o have in a secure cryptographic scheme

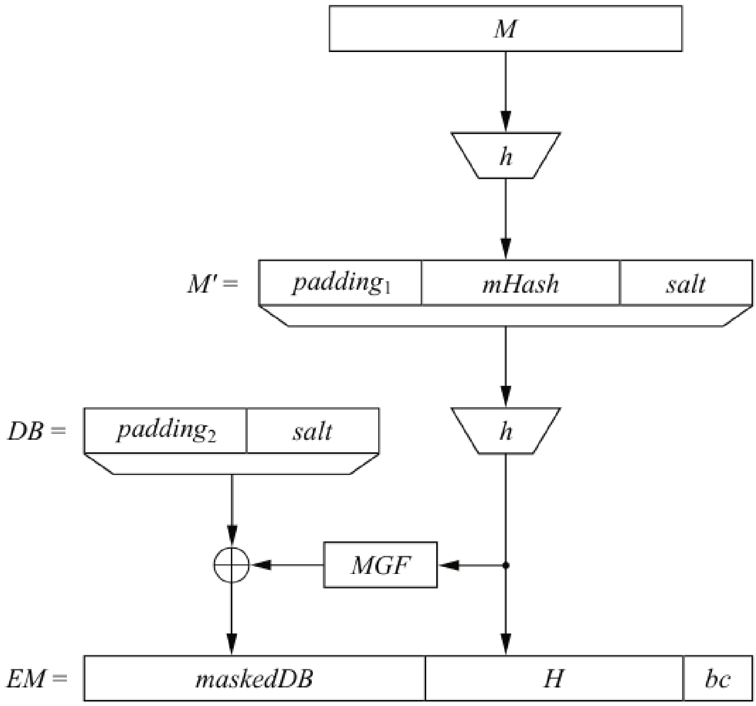
- **RSA Padding: The Probabilistic Signature Standard**
    - In this scheme, the message itself is not signed, but rather the hashed version of the message
    - *Encoding for the EMSA Probabilistic Signature Scheme*

      Let $|n|$ be the size of the RSA modulus in bits. The encoded message EM has a length $[(|n|-1)/8]$ bytes such that the bit length of EM is at most $|n|-1$ bits

        1. Generate a random value, *salt*
        2. Form a string, M' by concatenating a fixed padding, pad_1, the hash value, $mHash = h(M)$, and *salt*
        3. Compute a hash value H of the string M'
        4. Concatenate a fixed padding, pad_2 and *salt* to form a data block, DB
        5. Apply a mask generation function MGF to the string M' to compute the mask value $dbMask$
            - In practice, a hash function such as SHA-1 is often used
        6. XOR the mask value $dbMask$ and the data block DB to compute $maskedDB$

7. The encoded message EM is obtained by concatenating $maskedDB$, the hash value H, and the fixed padding bc

- After the encoding, the actual signature function is carried out with the encoded message and private key as inputs



10.3: The Elgamal Digital Signature Schemes

- **10.3.1: Schoolbook Elgamal Digital Signature**
  - Normal Elgamal key generation protocols will be followed such that:

$$k_{pub} = (p, \alpha, \beta)$$
$$k_{pr} = d$$

  - *Signature Generation*

1. Choose a random ephemeral key $k_E \in \{0, 1, 2, ..., p-1\}$ such that $gcd(k_E, p-1) = 1$

2. Compute the signature parameters:

$$r \equiv \alpha^{k_E} \mod p$$
$$s \equiv (x - d \cdot r)k_E^{-1} \mod p - 1$$

  - *Signature Verification*

1. Compute the value

$$t \equiv \beta^r \cdot r^s \bmod p$$

2. The verification compares t to $\alpha^x \bmod p$, and if they match validates the signature

10.4: The Digital Signature Algorithm (DSA)

- **10.4.1: The DSA Algorithm**
  - Here we will introduce DSA using the 1024 bit standard, but other longer bit lengths are also possible in the standard
  - *Key Generation*

    1. Generate a prime p with $2^{1023} < p < 2^{1024}$
    2. Find a prime divisor q of $p - 1$ with $2^{159} < q < 2^{260}$
    3. Find an element α with $ord(\alpha) = q$, i.e. α generates the subgroup with q elements
    4. Choose a random integer d between 0 and q
    5. Compute $\beta \equiv \alpha \bmod p$

    The keys are now:

    $$\begin{aligned} k_{pub} &= (p, q, \alpha, \beta) \\ k_{pr} &= (d) \end{aligned}$$

  - The central idea here is that there are two cyclic groups involved, namely the large cyclic group $\mathbb{Z}_p^*$ which has an order of bit length 1024, and its smaller subgroup which has an order of bit length 160
  - *Signature Generation*

    1. Choose an integer between 0 and q as the random ephemeral key, k_E
    2. Compute $r \equiv (\alpha^{k_E} \bmod p) \bmod q$
    3. Compute $s \equiv (SHA(x) + d \cdot r)k_E^{-1} \bmod q$

  - *Signature Verification*

1. Compute auxiliary value $w \equiv s^{-1} \bmod q$
2. Compute auxiliary value $u_1 \equiv w \cdot SHA(x) \bmod q$
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$
4. Compute $v \equiv (\alpha^{u_1} \cdot \beta^{u_2} \bmod p) \bmod q$
5. If v is equivalent to $r \bmod q$, then the signature is validated, otherwise it is not

10.5: The Elliptic Curve Digital Signature Algorithm (ECDSA)

- **10.5.1: The ECDSA Algorithm**
  - *Key Generation*
    1. Use an elliptic curve E with
       - modulus p
       - coefficients a and b
       - a point A which generates a cyclic group of prime order q
    2. Choose a random integer d between 0 and q
    3. Compute $B = dA$

    The keys are now:

$$k_{pub} = (p, a, b, q, A, B)$$
$$k_{pr} = (d)$$

  - *Signature Generation*
    1. Choose an integer between 0 and q as the random ephemeral key, k_E
    2. Compute $R = k_E A$
    3. Let $r = x_R$
    4. Compute $s \equiv (h(x) + d \cdot r) k_E^{-1} \bmod q$
  - *Signature Verification*
    1. Compute auxiliary value $w \equiv s^{-1} \bmod q$

2. Compute auxiliary value $u_1 \equiv w \cdot h(x) \bmod q$
3. Compute auxiliary value $u_2 \equiv w \cdot r \bmod q$
4. Compute $P = u_1 A + u_2 B$
5. If x_P (the x-coordinate of point P) is equivalent to $r \bmod q$ then the signature is validated