Justin Ciocoi Nov. 6, 2023

CSCI 360 Textbook Notes

Chapter 6: Intro to Public Key Cryptography

6.1: Symmetric vs. Asymmetric Cryptography

- Asymmetric, public-key algorithms are very different from symmetric algorithms such as AES or DES
- The symmetric algorithms that are widely in use today are very secure and fast, yet they do pose a number of disadvantages
    - *Key distribution problem*, such that sending the key over the secure channel is not possible given that the key is necessary to make the channel secure in the first place
    - *Number of Keys*, since for large symmetric networks, he number of keys can become a burden on the network capabilities
        - In a network with n users, the number of keys is equal to
        - $$\frac{n \cdot (n-1)}{2}$$
        - Additionally, every user has to securely store $n-1$ keys
    - Symmetric cryptography also does not provide any *nonrepudiation*, meaning a user cannot deny having sent a message
        - For example, since each user has the same private key, one of the users could fraudulently produce a message seemingly coming from the other
    - In public key cryptography each user has a public encryption key, k_{pub} which they publicly broadcast, as well as a matching private key, k_{priv}, which is used for decryption
    - This is shown in a very basic sense in the below diagram
    -

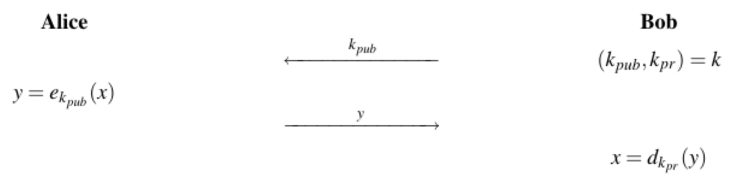


Fig. 6.4 Basic protocol for public-key encryption

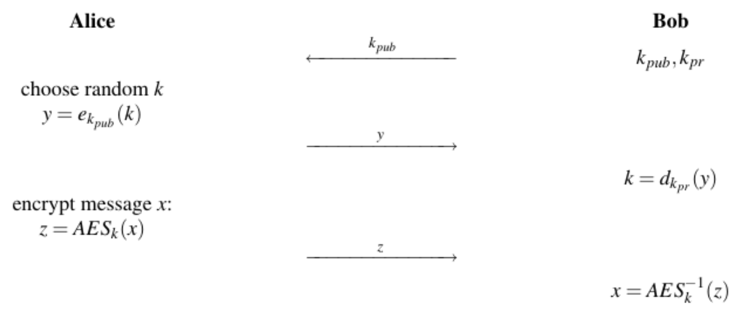
- The asymmetric protocol described here can also be modified in order to allow for symmetric key cryptography over an asymmetric protocol
  - Using the public-key algorithm, $y = e_{k_{pub}}(x)$, we can encrypt a symmetric key, like an AES key
  - Then, we can have the users securely exchange that symmetric key over the asymmetric protocol
  - Users can then use it to securely communicate
- This is shown below
- 

Fig. 6.5 Basic key transport protocol with AES as an example of a symmetric cipher

- **Definition 6.1.1:**
  - A function $f()$ is a *one-way* function if:
    - $y = f(x)$ is computationally easy
    - $x = f^{-1}(y)$ is computationally infeasible
- There are two popular one way functions which are used in practical public-key cryptography schemes
- The first is the integer factorization problem, on which RSA is based
- The other is the discrete logarithm function

6.2: Practical Aspects of Public-Key Cryptography

- **6.2.1: Security Mechanisms**
  - The main functions that public-key cryptography algorithms provide are
    - *Key Establishment*, such that a secret key can be established over insecure channels
    - *Nonrepudiation*, which is realized with digital signature algorithms
    - *Identification* using challenge-and-response protocols along with digital signatures
    - *Encryption* as outlined in the previous section
  - The major drawback of using public-key algorithms is that encryption is very computationally intensive, and thus extremely slow
    - Because of this, we often use symmetric cryptography to provide encryption and identification while asymmetric cryptography provides key establishment and nonrepudiation, resulting in a *hybrid network*
- **6.2.2: The Remaining Problem: Authenticity of Public Keys**
  - The problem of ensuring a particular public key belongs to a certain person can be solved using *certificates*, which, roughly speaking, bind public keys to certain identities
  - This is often used for e-commerce applications over the internet
- **6.2.3: Important Public-Key Algorithms**
  - In the world of public key cryptography, there are only three major families of public-key algorithm which are of practical relevance
  - These problems are all based on one of the three computational problems outlined below
    - *Integer Factorization Schemes*, which is based on the idea that it is difficult to factor large integers and is prominently used in RSA
    - *Discrete Logarithm Schemes*, which include the Diffie-Hellman key exchange or the Digital Signatures Algorithm
    - *Elliptical Curve (EC) Schemes*, which are a generalization of the discrete logarithm algorithm
- **6.2.4: Key Lengths and Security Levels**

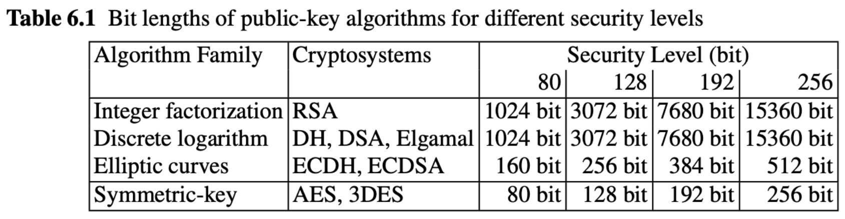
- Not surprisingly, the longer the operands and keys are in a public-key cryptographic scheme, the more secure the algorithms become
- An algorithm is said to have n bits of security when the best known attack on the algorithm requires 2^n operations
- In symmetric cryptography, a cipher with key length of 256 bits has 256 bits of security, but the situation with asymmetric cryptography is not quite as simple
- 

Table 6.1 Bit lengths of public-key algorithms for different security levels

Algorithm Family	Cryptosystems	Security Level (bit)			
		80	128	192	256
Integer factorization	RSA	1024 bit	3072 bit	7680 bit	15360 bit
Discrete logarithm	DH, DSA, Elgamal	1024 bit	3072 bit	7680 bit	15360 bit
Elliptic curves	ECDH, ECDSA	160 bit	256 bit	384 bit	512 bit
Symmetric-key	AES, 3DES	80 bit	128 bit	192 bit	256 bit

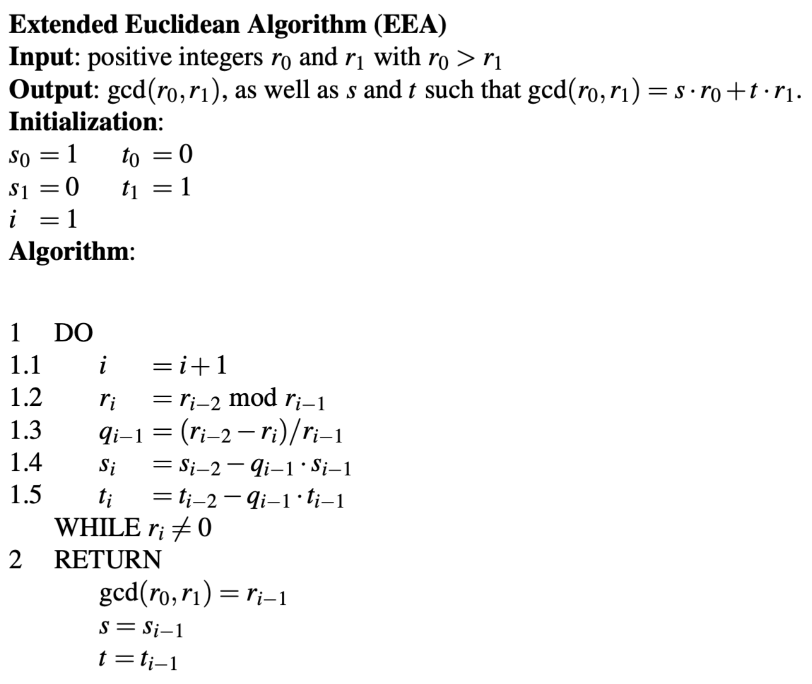
6.3: Essential Number Theory for Public Key Algorithms

- **6.3.1: Euclidean Algorithm**
  - We will begin with the problem of computing the greatest common divisor, or GCD, of two positive integers, r_0 and r_1, which is denoted by $gccd(r_0, r_1)$
  - For example, let $r_0 = 84$ and $r_1 = 30$
  - Factoring yields:
    - $r_0 = 84 = 2 * 2 * 3 * 7$
    - $r_1 = 30 = 2 * 3 * 5$
  - The GCD is the product of all common prime factors, so in this case
    - $gcd(30, 84) = 2 * 3 = 6$
  - Euclid's Algorithm posits that $gcd(r_0, r_1) = gcd(r_0 \ mod \ r_1, r_1)$
  - Let us now define Euclid's algorithm in a more formal fashion
    - **Input:** Positive integers r_0 and r_1 with $r_0 > r_1$
    - **Output:** $gcd(r_0, r_1)$
    - **Initialization:** $i = 1$
    - **Algorithm:**

        - Do
            - $i = i + 1$
            - $r_i = r_{i-2} \; mod \; r_{i-1}$
        - While $r_i \neq 0$
        - Return
            - $gcd(r_0, r_1) = r_{i-1}$

- **6.3.2: Extended Euclidean Algorithm**
    - In addition to computing the GCD of two integers, the extended Euclidean algorithm also computes a linear combination of the form
    - $$gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$$
    - Let us demonstrate with the problem $gcd(973, 301)$
        - $gcd(973, 301) = 973s + 301t$
        - $1 * 973 - 3 * 301 = 973 - 903 = 70$
        - $1 * 301 - 4 * 70 = 301 - 280 = 21$
            - $21 = 1 * 301 - 4 * (1 * 973 - 3 * 301)$
        - $1 * 70 - 3 * 21 = 70 - 63 = 7$
            - $7 = (973 - 3 * 301) - 3(301 - 4(973 - 3 * 301))$
            - let $973 = x, 301 = y$
            - $7 = (x - 3y) - 3(y - 4(x - 3y))$
                - $= (x - 3y) - 3(13y - 4x)$
                - $= (x - 3y) - 39y + 12x$
                - $7 = 13x - 42y$
    - Let us now try to find $12^{-1} \; mod \; 67$
        - $gcd(67, 12) = gcd(12, 7) = gcd(7, 5) = gcd(5, 2) = gcd(2, 1) = 1$

    - $gcd(67, 12) = 67s + 12t$
    - $1 * 67 - 5 * 12 = 7$
    - $1 * 12 - 1 * 7 = 5$
        - $5 = 1 * 12 - 1(1 * 67 - 5 * 12)$
    - $2 = 1 * 7 - 1 * 5$
        - $2 = (1a - 5b) - (6b - a)$
    - let $67 = a, 12 = b$
    - $1a - 5b = 7$
    - $1b - 1(1a - 5b) = 5$
        - $5 = 6b - a$
    - $2 = 2a - 11b$
    - $1 = 1 * 5 - 2 * 2$
        - $1 = (6b - a) - 2(2a - 11b)$
        - $1 = 28b - 5a$
        - $1 = 28(12) - 5(67)\ mod\ 67 = 28(12) - 0 = 1$
        - Thus, 28 is the inverse of 12 in mod 67
  - Below is the generalized algorithm for the extended Euclidean algorithm
  -



Extended Euclidean Algorithm (EEA)
Input: positive integers r_0 and r_1 with $r_0 > r_1$
Output: $\gcd(r_0, r_1)$, as well as s and t such that $\gcd(r_0, r_1) = s \cdot r_0 + t \cdot r_1$.
Initialization:

```
$s_0 = 1$    $t_0 = 0$
$s_1 = 0$    $t_1 = 1$
$i = 1$
```

Algorithm:

```
1     DO
1.1       $i = i + 1$
1.2       $r_i = r_{i-2} \bmod r_{i-1}$
1.3       $q_{i-1} = (r_{i-2} - r_i)/r_{i-1}$
1.4       $s_i = s_{i-2} - q_{i-1} \cdot s_{i-1}$
1.5       $t_i = t_{i-2} - q_{i-1} \cdot t_{i-1}$
      WHILE $r_i \neq 0$
2     RETURN
          $\gcd(r_0, r_1) = r_{i-1}$
          $s = s_{i-1}$
          $t = t_{i-1}$
```

- The main application of this algorithm in asymmetric cryptography is to compute the inverse modulo of an integer
- If we recall that a modular inverse exists if and only if $gcd(r_0, r_1) = 1$
  - Hence, when we apply the extended Euclidean algorithm, we get $s \cdot r_0 + t \cdot r_1$
  - When we take this equation in modulo r_0, we are left with
    - $$\begin{gathered} s \cdot r_0 + t \cdot r_1 = 1 \\ s \cdot 0 + t \cdot r_1 = 1 \ mod \ r_0 \\ r_1 \cdot t = 1 \ mod \ r_0 \end{gathered}$$
  - Which means that t is the inverse of r_1 in modulo r_0

- **6.3.3: Euler's Phi Function**
  - Let's now consider the ring $\mathbb{Z}_m$, or the set of integers $[0, 1, ..., m-1]$
  - We will be interested in the problem of figuring out *how many* numbers in this set are relatively prime to m
  - We can solve this problem using Euler's Phi function, which is defined below
    - **Definition 6.3.1:** *Euler's Phi Function*
    - The number of integers in $\mathbb{Z}_m$ relatively prime to m is denoted by
    - $$\Phi(m)$$

- Calculating Euler's phi function by running through all the elements and computing the GCD becomes extremely slow when the numbers are large, and completely infeasible when dealing with the magnitude of numbers involved in public-key cryptography
- Fortunately, we can calculate $\Phi(m)$ if we know the factorization of m, which is given in the following theorem
  - Given the following canonical factorization of m
    - $$m = p_1^{e_1} \cdot p_2^{e_2} \cdot \ldots \cdot p_n^{e_n}$$
  - where the p_i are distinct prime factors and e_i are positive integers, then
    - $$\Phi(m) = \prod_{i=1}^{n} (p_i^{e_i} - p_i^{e_i - 1})$$

- **6.3.4: Fermat's Little Theorem and Euler's Theorem**
  - *Fermat's Little Theorem* is a useful theorem in public-key cryptography that deals with primality testing and many other things
  - **Theorem 6.3.2:** *Fermat's Little Theorem*
    - Let a be an integer and p be a prime, then:
    - $$a^p = a \; mod \; p$$
    - The theorem can also be stated in this form
    - $$a^{p-1} \equiv 1 \; mod \; p$$
  - This theorem also gives us a simple way to find the modular inverse of a number
  - For an integer a, in mod p, $a^{-1} \equiv a^{p-2} \; mod \; p$
  - We can also generalize Fermat's Little Theorem to any integer modulo, not just primes
  - This generalization is *Euler's Theorem*
  - **Theorem 6.3.3:** *Euler's Theorem*
    - Let a and m be integers with $gcd(a, m) = 1$, then:
    - $$a^{\Phi(m)} \equiv 1 \; mod \; m$$