MAS 433: Cryptography

Lecture 18

Key Establishment and Management

Wu Hongjun

Lecture Outline

- Classical ciphers
- Symmetric key encryption
- Hash function and Message Authentication Code
- Public key encryption
- Digital signature
- **Key establishment and management**
  - Key generation
  - Key establishment
    - Key establishment using symmetric key cryptography
    - Key establishment using public key cryptography
      - PKI, Certificate: TLS/SSL
      - SSH
  - Secret sharing
    - Simple secret sharing
    - Shamir's threshold secret sharing scheme
    - Threshold public key cryptosystem
- Introduction to other cryptographic topics

Recommended Reading

- CTP: Section 10.5.3, 10.5.4
  Section 12.1,12.2,12.3,12.4,
  Section 13.1
- Wiki

  Key generation:
  - http://en.wikipedia.org/wiki/Random_number_generation
  - http://en.wikipedia.org/wiki//dev/random
  - http://en.wikipedia.org/wiki/CryptGenRandom
  - http://en.wikipedia.org/wiki/Hardware_random_number_generator

  Key establishment:
  - http://en.wikipedia.org/wiki/Kerberos_(protocol)
  - http://en.wikipedia.org/wiki/Public_key_certificate
  - http://en.wikipedia.org/wiki/Public_key_infrastructure
  - http://en.wikipedia.org/wiki/Transport_Layer_Security

  Secret Sharing:
  - http://en.wikipedia.org/wiki/Secret_sharing
  - http://en.wikipedia.org/wiki/Shamir%27s_Secret_Sharing
  - http://en.wikipedia.org/wiki/Threshold_cryptosystem

Key Generation

- **Secret** and **random** numbers
  - Important
    - used as the keys of symmetric key ciphers
    - used for generating the private keys of public key cryptosystems
    - used for padding in the RSA encryption scheme
    - used in the ElGamal encryption/digital signature, Digital Signature Algorithm
    - …….

Key Generation

- Security requirements for key generation
  - **Secret**
  - **Random**
    - Enough entropy/uncertainty should be introduced into the key generation process
  - **Independent** (at least computationally)
    - The disclosure of one random number should not affect the security of other random numbers

Key Generation

- **Serious mistake** in key generation
  - Use function "**random( )**" in Java or C programming for key generation
    - The function "random( )" generates random numbers from current time (in milliseconds)
      - Easy to guess time in milliseconds:

        2^{35} possible values in one year, not enough entropy
      - **Too weak for generating cryptographic keys**
    - Example 1
      - Several years ago, some local bank used the function "random( )" in Java to generate keys for symmetric ciphers
    - Example 2
      - In the early implementation of secure socket layer (SSL), the function "random()" in C is used to generate the symmetric keys

Key Generation

- Common approach to generate key
  - Step 1.
    - Get random number from entropy source
      - May not be uniformly distributed
    - Examples of source of entropy:
      - dice, coin flipping, roulette wheels
        - » Not convenient for cryptographic application
      - radioactive decay, thermal noise, clock drift, the timing of actual movements of a hard disk read/write head, and radio noise …..
      - Human use of system
        - » Mouse movements, timing of keystrokes …
  - Step 2.
    - Use randomness extractor to generate uniformly distributed random numbers from the random sequence from Step 1.
    - Example of random extractor: cryptographic hash function

Key Generation

- Strong random number generators
  - Software
    - /dev/random
      - Unix-like operating system (UNIX, Linux, …)
      - Collect information from the hard to predict events: keyboard and mouse use, network traffic packet timings, disk drive timings …
    - CryptGenRandom (or RtlGenRandom)
      - Microsoft Windows
      - Collect the following information:
        - The current process ID (GetCurrentProcessID).
        - The current thread ID (GetCurrentThreadID).
        - The tick count since boot time (GetTickCount).
        - The current time (GetLocalTime).
        - Various high-precision performance counters (QueryPerformanceCounter).
        - An MD4 hash of the user's environment block, which includes username, computer name, and search path. [...]
        - High-precision internal CPU counters, such as RDTSC, RDMSR, RDPMC
        - ……

Key Generation

- Strong random number generators (cont.)
  - Hardware
    - Intel 80802 Firmware Hub chip
      - Based on clock drift (the oscillators would be affected by thermal noise)
      - Not included in modern PC

- Problems:
  - Many entropy sources are often quite fragile, and fail silently, statistical tests on their output should be performed continuously.
  - Many random number generator devices include some statistical tests into the software that reads the device.

Key Establishment

- How to establish a secure channel (authenticated & encrypted) between any two users?
  - Pre-share secret keys between two users
    - Expensive
  - **Symmetric key cryptography (SKC) approach**
    - Each user shares secret keys with a trusted server
    - Then any two users can establish secret keys by communicating with the trusted server
  - **Public key cryptography (PKC) approach**
    - A user publishes its public key
    - Then others users can share secret keys with that user using public key cryptography technique

Key Establishment: SKC Approach

- How it works
  - Each user shares secret keys with a trusted server
  - Then any two users can establish secret keys by communicating with the trusted server
- Schemes
  - Kerberos
  - Bellare-Rogaway Scheme

Key Establishment: SKC: Kerberos

- Kerberos (animal)
  - Three-headed guard dog of Hades (god of the underworld in Greek mythology)

Key Establishment: SKC: Kerberos

- Kerberos (protocol/software)

  (not required for exam)

  – A popular session key distribution scheme
    - Mainly for authentication so as to access a service over the network
    - DES/AES is used in the scheme
  – Developed by MIT in the late 1980s

Key Establishment: SKC: Kerberos

- Simplified Kerberos V5 (not required for exam)

TA: trusted authority
Alice shares a secret key K_{Alice} with TA;
Bob shares a secret key K_{Bob} with TA;
Alice & Bob authenticate each other, and establish a secret key *K*

1. Alice chooses a random number, r_A. Alice sends $ID(Alice)$, $ID(Bob)$ and r_A to the TA.
2. The TA chooses a random session key K and a validity period (or *lifetime*), L. Then it computes a ticket to Bob,

$$t_{Bob} = e_{K_{Bob}}(K \parallel ID(Alice) \parallel L),$$

and

$$y_1 = e_{K_{Alice}}(r_A \parallel ID(Bob) \parallel K \parallel L).$$

The TA sends t_{Bob} and y_1 to Alice.

3. Alice decrypts y_1 using her key K_{Alice}, obtaining K. Then Alice determines the current time, $time$, and she computes

$$y_2 = e_K(ID(Alice) \parallel time).$$

Finally, Alice sends t_{Bob} and y_2 to Bob.

4. Bob decrypts t_{Bob} using his key K_{Bob}, obtaining K. He also decrypts y_2 using the key K, obtaining $time$. Then, Bob computes

$$y_3 = e_K(time + 1).$$

Finally, Bob sends y_3 to Alice.

Key Establishment: SKC: Kerberos

Simplified Kerberos V5 (cont.) (not required for exam)

1. When Alice decrypts y_1, she checks to see that the plaintext $d_{K_{Alice}}(y_1)$ has the form

$$d_{K_{Alice}}(y_1) = r_A \,||\, ID(Bob) \,||\, K \,||\, L,$$

for some K and L. If this condition does not hold, then Alice "rejects" and aborts the current session.

2. When Bob decrypts y_2 and t_{Bob}, he checks to see that the plaintext $d_K(y_2)$ has the form

$$d_K(y_2) = ID(Alice) \,||\, time$$

and the plaintext $d_{K_{Bob}}(t_{Bob})$ has the form

$$d_{K_{Bob}}(t_{Bob}) = K \,||\, ID(Alice) \,||\, L,$$

where $ID(Alice)$ is the same in both plaintexts and $time \leq L$. If these conditions hold, then Bob "accepts"; otherwise Bob "rejects."

3. When Alice decrypts y_3, she checks that $d_K(y_3) = time + 1$. If this condition holds, then Alice "accepts"; otherwise Alice "rejects."

Key Establishment: SKC: Kerberos

- Kerberos is complicated
  - **Message Authentication Code is not used** in Kerberos, so the authentication in Kerberos is somehow complicated (achieved through encryption/decryption)

Key Establishment: SKC: Bellare-Rogaway Scheme

- Bellare-Rogaway key establishment scheme (1995)

1. Alice chooses a random number, r_A, and she sends $ID(Alice)$, $ID(Bob)$ and r_A to Bob.
2. Bob chooses a random number, r_B, and he sends $ID(Alice)$, $ID(Bob)$, r_A and r_B to the TA.
3. The TA chooses a random session key K. Then it computes

$$y_B = (e_{K_{Bob}}(K), MAC_{Bob}(ID(Alice) \,||\, ID(Bob) \,||\, r_B \,||\, e_{K_{Bob}}(K)))$$

and

$$y_A = (e_{K_{Alice}}(K), MAC_{Alice}(ID(Bob) \,||\, ID(Alice) \,||\, r_A \,||\, e_{K_{Alice}}(K))).$$

The TA sends y_B to Bob and y_A to Alice.

Key Establishment: PKC Approach

- Key Establishment using PKC
  - Method 1: Public key encryption schemes
    - RSA, ElGamal …
  - Method 2: Diffie-Hellman Key exchange (1976)

Key Establishment: PKC: DH Key Exchange

- Diffie-Hellman Key exchange (1976)
  - The first public key cryptosystem
  - Based on discrete logarithm
  - The algorithm
    - Alice and Bob agree on a cyclic group G and a generator g of G
    - Alice selects a random number a and sends $y_A = g^a \bmod p$ to Bob

      Bob selects a random number b and sends $y_B = g^b \bmod p$ to Alice
    - Alice computes the key as $K = y_B^a \bmod p$

      Bob computes the key as $K = y_A^b \bmod p$

      The shared secret key is $K = g^{ab} \bmod p$

Key Establishment: PKC Approach

- An example of using public encryption scheme to establish secret key
  - SSH (Secure Shell)
    - Used for remote secure access to Linux/Unix system
    - Key establishment:
      - Step 1. A user receives a public key from the Linux/UNIX server
      - Step 2. **If the user trusts the received public key**, then types "yes", and the public key is used to establish a secret key between the user's computer and the Linux/UNIX system
        - » Later, the trusted public key is stored on the user's computer, and no confirmation is needed for future accesses

Key Establishment: PKC Approach

- Sending public key over internet
  - **Secure**
    - If the attacker does not have the capability to modify the internet traffic
  - **Insecure**
    - If the attacker has the ability to modify the internet traffic
      - The attacker can launch the man-in-the-middle attack**:**

Suppose that Alice is sending her public key to Bob

1) The attacker Eve can modify the public key of Alice to Eve's public key

2) Bob uses Alice's public key to establish secret key with Alice. But if Alice's public key has been changed to Eve's public key, Bob would use Eve's public key to establish a secret key with Eve

3) Eve establishes a secret key with Alice using the public key of Alice

4) Eve receives the encrypted and authenticated data from Bob, then decrypts it, and transmits it securely to Alice; Eve receives the encrypted and authenticated data from Alice, decrypts it, and transmits it securely to Alice.

In this way, Eve can decrypt all the communication between Alice and Bob, but Alice and Bob do not notice the existence of Eve.

Key Establishment: PKC Approach

- How to ensure the authenticity of public key?
  - Personal exchange, trusted courier ….
    - Inconvenient to use
    - But useful for certain applications
      - Such as delivering the government's public key that is used to sign the e-passport
  - **Online** trusted server
    - The trusted server stores the public keys of all users
    - Everyone is assumed to know the public key (for signature) of that server
    - Then everyone can establish a secure (authenticated) channel with the server to retrieve the public key of other user
    - But the server must be kept online
  - **Offline** trusted authority
    - Normally **Certification authority** (CA) and **public key certificate** are involved
    - Everyone knows the public key (for signature) of CA
    - A user registers its public key with CA. CA signs {the user's information + the user's public key}, and give the signature (certificate) to the user.
    - A public key is sent together with the certificate for verification (using CA's public key)
    - **The commonly used approach**

Key Establishment: PKC: Public Key Certificate

- A certificate binds a public key with respective user identitiy
- Signed by a certificate authority (CA) or other trusted party

- Public key infrastructure (PKI)
  - create, manage, distribute, use, store, and revoke digital certificates

Key Establishment: PKC: Public Key Certificate

- Public key certificate
  - **X.509**
    - widely used
      - IPSec, TLS/SSL ….
    - Now version 3
  - PGP
    - Used in PGP email protection software
  - …..

Key Establishment: PKC: Public Key Certificate

Contents of X.509 digital certificate:

- Version
- Serial number
- Signature algorithm ID
- Issuer
- Validity
  - Not Before
  - Not After
- Subject
- Subject public key info
  - Public key algorithm
  - Subject public key
- (optional information)
- Certificate signature

Version: v1, v2 or v3
Serial number: Used to uniquely identify the certificate.
Signature algorithm ID: the algorithm used to create the signature
Issuer: (CA) The entity that verified the information and issued the certificate.
Valid-from: The date the certificate is first valid from.
Valid-to: The expiration date.
Subject: The person, or entity identified.
Subject public key info: the public key of the subject
Certificate signature: authenticate all the above contents, signed by the issuer (CA)

X.509 certificate example

```
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number: 7829 (0x1e95)
        Signature Algorithm: md5WithRSAEncryption
        Issuer: C=ZA, ST=Western Cape, L=Cape Town, O=Thawte Consulting cc,
                OU=Certification Services Division,
                CN=Thawte Server CA/emailAddress=server-certs@thawte.com
        Validity
            Not Before: Jul  9 16:04:02 1998 GMT
            Not After : Jul  9 16:04:02 1999 GMT
        Subject: C=US, ST=Maryland, L=Pasadena, O=Brent Baccala,
                 OU=FreeSoft, CN=www.freesoft.org/emailAddress=baccala@freesoft.org
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
            RSA Public Key: (1024 bit)
                Modulus (1024 bit):
                    00:b4:31:98:0a:c4:bc:62:c1:88:aa:dc:b0:c8:bb:
                    33:35:19:d5:0c:64:b9:3d:41:b2:96:fc:f3:31:e1:
                    66:36:d0:8e:56:12:44:ba:75:eb:e8:1c:9c:5b:66:
                    70:33:52:14:c9:ec:4f:91:51:70:39:de:53:85:17:
                    16:94:6e:ee:f4:d5:6f:d5:ca:b3:47:5e:1b:0c:7b:
                    c5:cc:2b:6b:c1:90:c3:16:31:0d:bf:7a:c7:47:77:
                    8f:a0:21:c7:4c:d0:16:65:00:c1:0f:d7:b8:80:e3:
                    d2:75:6b:c1:ea:9e:5c:5c:ea:7d:c1:a1:10:bc:b8:
                    e8:35:1c:9e:27:52:7e:41:8f
                Exponent: 65537 (0x10001)
    Signature Algorithm: md5WithRSAEncryption
        93:5f:8f:5f:c5:af:bf:0a:ab:a5:6d:fb:24:5f:b6:59:5d:9d:
        92:2e:4a:1b:8b:ac:7d:99:17:5d:cd:19:f6:ad:ef:63:2f:92:
        ab:2f:4b:cf:0a:13:90:ee:2c:0e:43:03:be:f6:ea:8e:9c:67:
        d0:a2:40:03:f7:ef:6a:15:09:79:a9:46:ed:b7:16:1b:41:72:
        0d:19:aa:ad:dd:9a:df:ab:97:50:65:f5:5e:85:a6:ef:19:d1:
        5a:de:9d:ea:63:cd:cb:cc:6d:5d:01:85:b5:6d:c8:f3:d9:f7:
        8f:0e:fc:ba:1f:34:e9:96:6e:6c:cf:f2:ef:9b:bf:de:b5:22:
        68:9f
```

Key Establishment: PKC: Public Key Certificate

- Verification of certificate
  - Within validity period?
  - Verify the signature using the public key of CA
  - Certificate revoked?
    - Certificates can be revoked before expiration if user's private key or CA's private key are compromised
    - CA maintains a Certificate Revocation List (CRL)
    - Users need to check CA's CRL

Secret Sharing

- Secret sharing
  - Distribute a secret among a group of participants, each of which is allocated a share of the secret.
  - The secret can be reconstructed only when a sufficient number of shares are combined together; individual shares are of no use on their own

Secret Sharing

- Threshold secret sharing
  - Distribute a secret among n participants
  - The secret can be reconstructed only when **at least t** shares are combined together
    - t-out-of-n secret sharing scheme
    - Alternatively, (t,n) secret sharing scheme
  - Application example:
    - In the early 1990s in Russia, control of nuclear weapons depended upon a two-out-of-three access mechanism
      - Three parties: president, defense minister, defense ministry
      - Any two parties can control nuclear weapons

Secret Sharing: (n, n) secret sharing

- n-out-of-n secret sharing
  - The simplest threshold secret sharing
  - Distribute a secret among n participants
  - The secret can be reconstructed only when all the shares are combined together

Secret Sharing: (n, n) secret sharing

- The n-out-of-n secret sharing scheme
  - Let the secret be encoded as an integer S
  - Generate n-1 random number r_i $(1 \leq i \leq n\text{-}1)$, where each r_i is the same size as that of S
  - Let $r_n = S \oplus r_1 \oplus r_2 \oplus r_3 \oplus \ldots \oplus r_{n\text{-}1}$
  - r_i $(1 \leq i \leq n)$ are the n shares of the secret s
    - Reconstructing S requires all the n shares
    - $S = r_1 \oplus r_2 \oplus r_3 \oplus \ldots \oplus r_{n\text{-}1} \oplus r_n$

Secret Sharing: (n, n) secret sharing

- Security
  - Unconditionally secure
    - With less than n shares, no information of S can be recovered

Secret Sharing: (n, n) secret sharing

- Insecure n-out-of-n secret sharing scheme
  - Suppose that a secret key K is to be shared among n participants
  - Divide K into small pieces (K is αn-bit, each k_i is α-bit)

    $$K = k_1 \| k_2 \| k_3 \| \ldots \| k_n$$
  - The i-th participant receives k_i
- Attack
  - With t shares, $t \cdot \alpha$ bits of K are known
    - Then recovering K requires only $2^{(n-t)\cdot\alpha}$ computations, instead of $2^{n\cdot\alpha}$ computations (less than n shares can be used to reconstruct the secret)

Secret Sharing:
Shamir's Secret Sharing Scheme

- Shamir's secret sharing scheme
  - (t, n) secret sharing scheme
  - Based on simple math
- Basic idea
  - 2 points are sufficient to define a line
  - 3 points are sufficient to define a parabola
  - 4 points to define a cubic curve

=> it takes t points to define a polynomial of degree t-1

Secret Sharing:
Shamir's Secret Sharing Scheme

- Shamir's scheme over finite field GF(p)
  (p is a large prime, the secret S is an integer less than p)
  - Generate t-1 random integers a_i (i = 1, 2, 3, …, t-1)
    (each a_i is an integer less than p)
  - Let $a_0 = S$
  - Build the polynomial over GF(p):
    $$f(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \ldots + a_{t-1}x^{t-1}$$
  - Compute n points: (j, $f(j)$) for j = 1, 2, 3, …, n
  - Each participant is given a point (a share)
- Reconstructing the secret S
  - Given t points, solve the linear equations to determine the coefficients of the polynomial
  - $S = a_0$

Secret Sharing:
Shamir's Secret Sharing Scheme

- Use Lagrange interpolation to find the coefficient a_0 efficiently
- Lagrange polynomial (not required for exam)
  - Given t points (x_j, y_j) of a polynomial with degree t-1, the interpolation polynomial in the Lagrange form can be written as

$$L(x) = \sum_{j=1}^{t} y_j \lambda_j(x), \text{ where } \lambda_j(x) = \left(\prod_{\substack{1 \le i \le t \\ i \ne j}} \frac{x - x_i}{x_j - x_i}\right) \bmod p$$

- The coefficient a_0 is given as

$$a_0 = L(0) = \left(\sum_{j=1}^{t} y_j \lambda_j(0)\right) \bmod p$$

Secret Sharing:
Shamir's Secret Sharing Scheme

- Security
  - Unconditionally secure
    - With less than t shares (points), no information of S can be recovered

Secret Sharing: Threshold public key cryptosystem

- In a public key cryptosystem, the simple (n, n) secret sharing scheme and Shamir's (t, n) secret sharing scheme can be used to share a private key, and the private key can be successfully reconstructed if sufficient number of shares are given
  - But we do not want to reconstruct the private key
  - How to design such a threshold public key cryptosystem?

Secret Sharing: Threshold public key cryptosystem

- (n,n) threshold public key cryptosystem
  - Example: (3,3) threshold ElGamal encryption scheme
    - Let the private key $x = x_1+x_2+x_3 \bmod p\text{-}1$, where x_1, x_2 are random integers
    - x_1,x_2, x_3 are given to three participants A_1, A_2 and A_3, respectively
    - After receiving a ciphertext (c_1,c_2),
      - A_1 computes $w_1 = (c_1)^{x1} \bmod p$
      - A_2 computes $w_2 = (c_1)^{x2} \bmod p$
      - A_3 computes $w_3 = (c_1)^{x3} \bmod p$
      - The message is decrypted as: $(w_1 \cdot w_2 \cdot w_3)^{-1} \cdot c_2 \bmod p = m$
        (the message is decrypted without each participant revealing its share x_i to others)

Secret Sharing: Threshold public key cryptosystem

- (t, n) threshold public key cryptosystem
  - A Simple Example: (2, 3) threshold ElGamal encryption scheme
    - Let the private key $x = x_1+x_2+x_3 \bmod p\text{-}1$, where x_1, x_2 are random numbers
    - x_1, x_2 are given to participant A_1, respectively
      x_2, x_3 are given to participant A_2, respectively
      x_1, x_3 are given to participant A_3, respectively

    - After receiving ciphertext (c_1,c_2),
      - If only A1 and A2 are available, then
        - » A_1 computes $w_1 = (c_1)^{x1} \bmod p$
        - » A_2 computes $w_2 = (c_1)^{x2} \bmod p$; $w_3 = (c_1)^{x3} \bmod p$
        - » The message is decrypted as: $(w_1 \cdot w_2 \cdot w_3)^{-1} \cdot c_2 \bmod p = m$
      - If only A1 and A3 are available, then
        - » A_1 computes $w_1 = (c_1)^{x1} \bmod p$, $w_2 = (c_1)^{x2} \bmod p$
        - » A_3 computes $w_3 = (c_1)^{x3} \bmod p$
        - » The message is decrypted as: $(w_1 \cdot w_2 \cdot w_3)^{-1} \cdot c_2 \bmod p = m$
      - If only A2 and A3 are available, then
        - » A_2 computes $w_2 = (c_1)^{x2} \bmod p$, $w_3 = (c_1)^{x3} \bmod p$
        - » A_3 computes; $w_1 = (c_1)^{x1} \bmod p$
        - » The message is decrypted as: $(w_1 \cdot w_2 \cdot w_3)^{-1} \cdot c_2 \bmod p = m$

Secret Sharing: Threshold public key cryptosystem

- (t, n) threshold public key cryptosystem based on Shamir's secret sharing scheme

  (not required for exam)
  - Example: Threshold ElGamal encryption scheme
    - Consider a slightly modified ElGamal encryption scheme: the large prime p satisfies p-1= 2q, q is a prime, and the order of g is q
    - The private key x is shared using Shamir's secret sharing scheme over GF(***q***)
      - Generate t-1 random integers a_i (i = 1, 2, 3, …, t-1)
        (each a_i is an integer less than q)
      - Let $a_0 = S$
      - Build the polynomial over GF(q):
        $f(z) = a_0 + a_1 z + a_2 z^2 + a_3 z^3 + \ldots + a_{t-1} z^{t-1}$
      - Compute n points: $(u_j, z_j) = (j, f(j))$ for j = 1, 2, 3, …, n
      - Each participant is given a point (u_j, z_j)

Secret Sharing: Threshold public key cryptosystem

- (t, n) threshold public key cryptosystem based on Shamir's secret sharing scheme

  (not required for exam)

  - Example: Threshold ElGamal encryption scheme (cont.)
    - After receiving a ciphertext (c_1, c_2), it is decrypted by t participants as follows:
      - Each participant computes $w_j = (c_1)^{z_j} \bmod p$ for $1 \le j \le t$
      - Compute $\lambda_j(0) = \prod_{\substack{1 \le i \le t \\ i \ne j}} \frac{0 - u_i}{u_j - u_i} \bmod q$ (public information)
      - Then the message is obtained as $(\prod_{j=1}^{t} (w_j)^{\lambda_j(0)})^{-1} c_2 \bmod p$

Proof. $\prod_{j=1}^{t} (w_j)^{\lambda_j(0)} \bmod p = \prod_{j=1}^{t} (c_1)^{z_j \cdot \lambda_j(0)} \bmod p = (c_1)^{\sum_{j=1}^{t} z_j \cdot \lambda_j(0)} \bmod p$

$= (c_1)^{a_0} \bmod p = (c_1)^{x} \bmod p$

Summary

- Key generation
  - Good entropy source is needed
    - Avoid using the function “random( )” function to generate key
  - Apply randomness extractor to enhance randomness
- Key establishment
  - Key establishment using symmetric key cryptography
    - Kerberos
    - Bellare-Rogaway key establishment scheme
  - Key establishment using public key cryptography
    - SSH
    - PKI, public key certificate: authenticate public keys
      - TLS/SSL
- Secret sharing
  - (n, n) secret sharing
  - Shamir’s secret sharing scheme
  - Threshold public key cryptosystem
    - (n, n) threshold public key cryptosystem
    - (t, n) threshold public key cryptosystem
    - (t, n) threshold ElGamal encryption scheme based on Shamir’s secret sharing scheme