密钥管理理论及基础

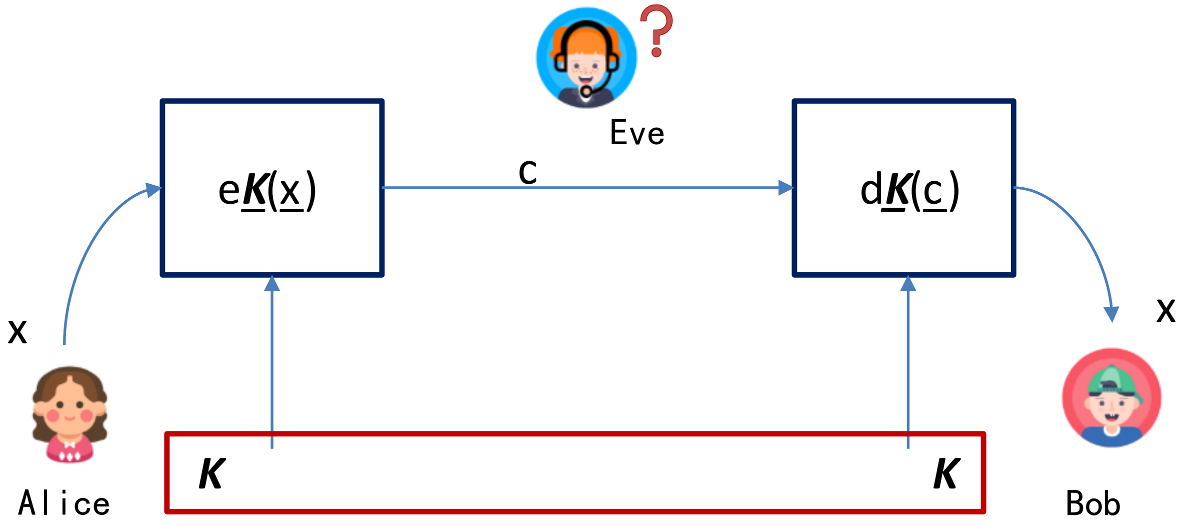
开源开发实践-第五周

David & Cary

CONTENT
目 录

Public-key Cryptography

Secret-key vs. Public-key



Public-key Cryptography

Secret-key vs. Public-key

- In the classical cryptography, Alice and Bob secretly choose a key K and an encryption rule ***eK*** and a decryption rule ***dK***, where ***dK*** is either the same as ***eK***, or derived from it. A cryptosystem like this is known as a **secret-key** cryptosystem or, alternatively, a **symmetric-key** cryptosystem.
- One drawback of a secret-key system is that it requires the prior communication of the key K between Alice and Bob, using a secure channel, before any ciphertext is transmitted.
- The idea behind a **public-key** cryptosystem or, alternatively, a **asymmetric-key** cryptosystem is that it might be possible to find a cryptosystem where it is computationally infeasible to determine ***dK*** given ***eK***.
- encryption rule ***eK*** is a public key, the value of which can be made known to everyone, hence it is called as public-key system.
- Bob will be the only person that can decrypt the ciphertext, using the decryption rule ***dK***, which is called the private key.

Public-key Cryptography

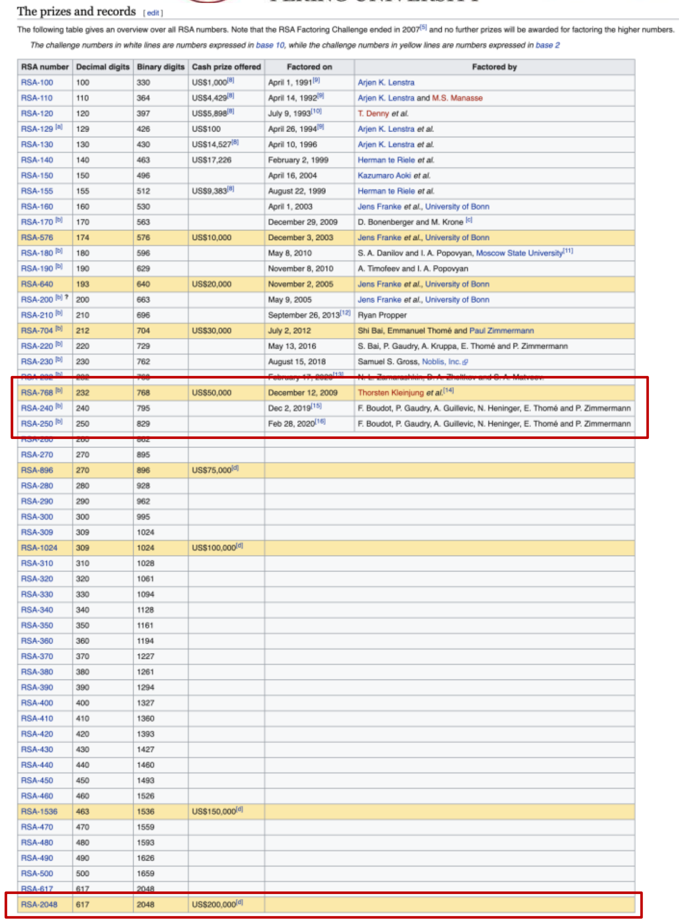
Why Public-key?

- Developed to address two main issues
  - ✓ Key distribution
  - ✓ Digital signatures
- Invented by Diffie and Hellman in 1976
- The main idea behind public-key cryptosystem is the concept of the trapdoor one-way function
  - ✓ Easy: $x \Rightarrow y,\ y = f(x)$
  - ✓ **Difficult** : $y \Rightarrow x,\ x = f^{-1}(y)$
  - ✓ Easy: $y \Rightarrow x,\ x = f^{-1}(y)$, if use **trapdoor** as the private key

Public-key Cryptography

RSA Cryptosystem and Factoring Integers

- One of the most important and widely used Public-key Cryptosystems is RSA (**R**ivest, **S**hamir & **A**dleman), in which the security is based on **the difficulty of factoring large integers**.
- Example, when n is large, n = p × q is a one-way function.
  - ✓ **Easy,** given p and q ➔ calculate n
  - ✓ **Difficult,** given n ➔ calculate p and q
- This is the factorization problem.
- A 768-bit RSA prime number:
  1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413
- In 2001, RSA Laboratories expanded the factoring challenge and offered prizes ranging from $10,000 to $200,000 for factoring numbers from 576 bits up to 2048 bits



The prizes and records [edit]

The following table gives an overview over all RSA numbers. Note that the RSA Factoring Challenge ended in 2007[5] and no further prizes will be awarded for factoring the higher numbers.

The challenge numbers in white lines are numbers expressed in base 10, while the challenge numbers in yellow lines are numbers expressed in base 2

RSA number	Decimal digits	Binary digits	Cash prize offered	Factored on	Factored by
RSA-100	100	330	US$1,000[8]	April 1, 1991[9]	Arjen K. Lenstra
RSA-110	110	364	US$4,429[8]	April 14, 1992[9]	Arjen K. Lenstra and M.S. Manasse
RSA-120	120	397	US$5,898[8]	July 9, 1993[10]	T. Denny *et al.*
RSA-129 [a]	129	426	US$100	April 26, 1994[9]	Arjen K. Lenstra *et al.*
RSA-130	130	430	US$14,527[8]	April 10, 1996	Arjen K. Lenstra *et al.*
RSA-140	140	463	US$17,226	February 2, 1999	Herman te Riele *et al.*
RSA-150	150	496		April 16, 2004	Kazumaro Aoki *et al.*
RSA-155	155	512	US$9,383[8]	August 22, 1999	Herman te Riele *et al.*
RSA-160	160	530		April 1, 2003	Jens Franke *et al.*, University of Bonn
RSA-170 [b]	170	563		December 29, 2009	D. Bonenberger and M. Krone [c]
RSA-576	174	576	US$10,000	December 3, 2003	Jens Franke *et al.*, University of Bonn
RSA-180 [b]	180	596		May 8, 2010	S. A. Danilov and I. A. Popovyan, Moscow State University[11]
RSA-190 [b]	190	629		November 8, 2010	A. Timofeev and I. A. Popovyan
RSA-640	193	640	US$20,000	November 2, 2005	Jens Franke *et al.*, University of Bonn
RSA-200 [b] ?	200	663		May 9, 2005	Jens Franke *et al.*, University of Bonn
RSA-210 [b]	210	696		September 26, 2013[12]	Ryan Propper
RSA-704 [b]	212	704	US$30,000	July 2, 2012	Shi Bai, Emmanuel Thomé and Paul Zimmermann
RSA-220 [b]	220	729		May 13, 2016	S. Bai, P. Gaudry, A. Kruppa, E. Thomé and P. Zimmermann
RSA-230 [b]	230	762		August 15, 2018	Samuel S. Gross, Noblis, Inc.
RSA-232 [b]	232	768		February 17, 2020[13]	N. L. Zamarashkin, D. A. Zheltkov and S. A. Matveev.
RSA-768 [b]	232	768	US$50,000	December 12, 2009	Thorsten Kleinjung *et al.*[14]
RSA-240 [b]	240	795		Dec 2, 2019[15]	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P. Zimmermann
RSA-250 [b]	250	829		Feb 28, 2020[16]	F. Boudot, P. Gaudry, A. Guillevic, N. Heninger, E. Thomé and P. Zimmermann
RSA-260	260	862			
RSA-270	270	895			
RSA-896	270	896	US$75,000[d]		
RSA-280	280	928			
RSA-290	290	962			
RSA-300	300	995			
RSA-309	309	1024			
RSA-1024	309	1024	US$100,000[d]		
RSA-310	310	1028			
RSA-320	320	1061			
RSA-330	330	1094			
RSA-340	340	1128			
RSA-350	350	1161			
RSA-360	360	1194			
RSA-370	370	1227			
RSA-380	380	1261			
RSA-390	390	1294			
RSA-400	400	1327			
RSA-410	410	1360			
RSA-420	420	1393			
RSA-430	430	1427			
RSA-440	440	1460			
RSA-450	450	1493			
RSA-460	460	1526			
RSA-1536	463	1536	US$150,000[d]		
RSA-470	470	1559			
RSA-480	480	1593			
RSA-490	490	1626			
RSA-500	500	1659			
RSA-617	617	2048			
RSA-2048	617	2048	US$200,000[d]		

https://en.wikipedia.org/wiki/RSA_Factoring_Challenge

Public-key Cryptography

RSA Cryptosystem and Factoring Integers

- Before describing how the RSA Cryptosystem works, we need to discuss some more facts concerning modular arithmetic and number theory. Basically, two fundamental tools are required:
- The Euclidean Algorithm
- The Chinese Remainder Theorem

Public-key Cryptography

RSA Cryptosystem and Factoring Integers

- The **Euclidean algorithm** is based on the principle that the **greatest common divisor** of two numbers does not change if the larger number is replaced by its difference with the smaller number.
- For example, 21 is the GCD of 252 and 105 (as 252 = 21 × 12 and 105 = 21 × 5), and the same number 21 is also the GCD of 105 and 252 − 105 = 147.
- Since this replacement reduces the larger of the two numbers, repeating this process gives successively smaller pairs of numbers until the two numbers become equal. When that occurs, they are the GCD of the original two numbers.
- This is an efficient method for computing the greatest common divisor (GCD) of two integers, the largest number that divides them both without a remainder.
- It can be used to determine if a positive integer $b < n$ has a multiplicative inverse modulo n, by calling Euclidean Algorithm(n, b) and checking to see if $r_m = 1$.
- https://asecuritysite.com/encryption/inve

Public-key Cryptography

RSA Cryptosystem and Factoring Integers

- The **Chinese Remainder Theorem**
- A problem described in an ancient Chinese arithmetic book（中国东汉时期的《九章算术》）.
- Problem: We have a number of things, but we do not know exactly how many. If we count them by threes we have two left over. If we count them by fives we have three left over. If we count them by sevens we have two left over. How many things are there?
- X = 2mod3, X = 3mod5, X = 2mod7, X = ?
- For efficiency, many popular crypto libraries (such as OpenSSL, Java and .NET) use an optimized algorithm for decryption and signing based on the Chinese remainder theorem.

Public-key Cryptography

RSA Cryptosystem and Factoring Integers

- RSA Parameters Generation

1. Generate two large primes, p and q, such that p != q
2. $n \leftarrow p\,q$ and $\phi(n) \leftarrow (p-1)(q-1)$
3. Choose a random b ($1 < b < \phi(n)$) such that $\gcd(b, \phi(n)) = 1$
4. $a \leftarrow b^{-1} \bmod \phi(n)$
5. The public key is **(n, b)** and the private key is **(p, q, a)**.

Public-key Cryptography

RSA Cryptosystem and Factoring Integers

- Example, suppose Bob chooses

1. p = 101 and q = 113
2. n = 11413 and ϕ(n) = 100 × 112 = 11200
3. Since 11200 = $2^6 5^2 7$, an integer b can be used as an encryption exponent if and only if b is not divisible by 2, 5, or 7. In practice, however, Bob will not factor ϕ(n). He will verify that gcd(ϕ(n), b) = 1 using CRT. Suppose Bob chooses b = 3533.
4. then b^{-1} mod 11200 = 6597. So, Bob's secret decryption exponent is a = 6597.
5. Bob publishes **n = 11413** and **b = 3533** as his public key, and hold **p = 101**, **q = 103**, and **a = 6597** as private key.

- Now, suppose Alice wants to encrypt the plaintext 9726 to send to Bob.

1. Alice will compute 9726^{3533} mod **11413** = 5761, and send the ciphertext 5761 to Bob.
2. Bob uses his secret decryption exponent to compute 5761^{6597} mod **11413** = 9726.

Public-key Cryptography

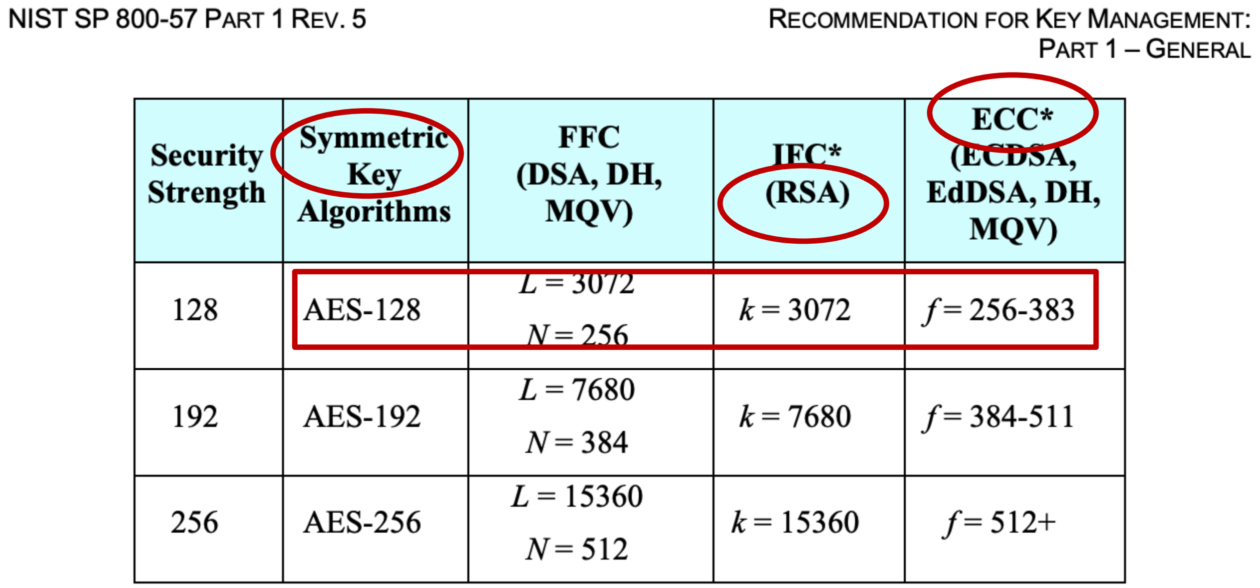
Public-Key Cryptography and Discrete Logarithms

- ElGamal Public-key Cryptosystem
- Example, when n is large, the function y = x^k mod n is a trapdoor one-way function.
  - ✓ **Easy,** given x, k, and n ➔ calculate y
  - ✓ **Difficult,** given y, k, and n ➔ calculate x
- This is the discrete logarithm problem (离散对数问题).
- However, if we know the trapdoor, k′, such that k × k′ = 1 mod f(n), we can use x = yk′ mod n to find x.

Public-key Cryptography

Security Strength

- Recommendation for Key Management by NIST (National Institute of Standards and Technology), May 2020



NIST SP 800-57 PART 1 REV. 5

RECOMMENDATION FOR KEY MANAGEMENT: PART 1 – GENERAL

Security Strength	Symmetric Key Algorithms	FFC (DSA, DH, MQV)	IFC* (RSA)	ECC* (ECDSA, EdDSA, DH, MQV)
128	AES-128	$L = 3072$ $N = 256$	$k = 3072$	$f = 256\text{-}383$
192	AES-192	$L = 7680$ $N = 384$	$k = 7680$	$f = 384\text{-}511$
256	AES-256	$L = 15360$ $N = 512$	$k = 15360$	$f = 512+$

https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf

Public-key Cryptography

Key Establishment

- Most **public-key** cryptosystems are typically much slower than commonly-used **secret-key** cryptosystems.
- Secret-key cryptosystems are used to encrypt "long" messages while public-key cryptosystems are mainly used to encrypt "short" keys used in secret-key cryptosystems.
- Three approaches to establish secret keys:
  - ✓ Key Pre-distribution: A trusted authority (TA) distributes keying information ahead of time in a secure channel to users.
  - ✓ Session Key Distribution: A key distribution center (KDC) chooses session keys and distributes them to users via an interactive session key distribution scheme.
  - ✓ Key Agreement: Users employ an interactive key agreement scheme to negotiate a session key over a public channel.
- Questions: how to establish secret keys for PG KMS?

Public-key Cryptography

long-lived keys vs. session keys

- Long-lived keys

  Users may have long-lived keys that are pre-computed and then stored securely. long-lived keys could be secret keys known to a pair of users or to a user and the TA. On the other hand, they could be private keys corresponding to public keys.

- Session keys

  Pairs of users will often employ secret short-lived session keys in a particular session, and then throw them away when the session has ended. Session keys are usually secret keys, for use in a secret-key cryptosystem.

- Question: should we use long-lived keys or session keys or both for PG KMS?

Public-key Cryptography

Key Distribution

- Diffie-Hellman Key distribution
- each party generates a public/private key pair and distributes the public key. After obtaining an authentic copy of each other's public keys, Alice and Bob can compute a shared secret offline. The shared secret can be used, for instance, as the key for a symmetric cipher.

Public-key Cryptography

Diffie-Hellman Key distribution

Alice

Known	Unknown
$p = 23$	
$g = 5$	
$a = 6$	b
$A = 5^6 \bmod 23 = 8$	
$B = 19$	
$s = 19^6 \bmod 23 = 2$	

Eve

Known	Unknown
$p = 23$	
$g = 5$	
	a, b
$A = 8, B = 19$	
	s

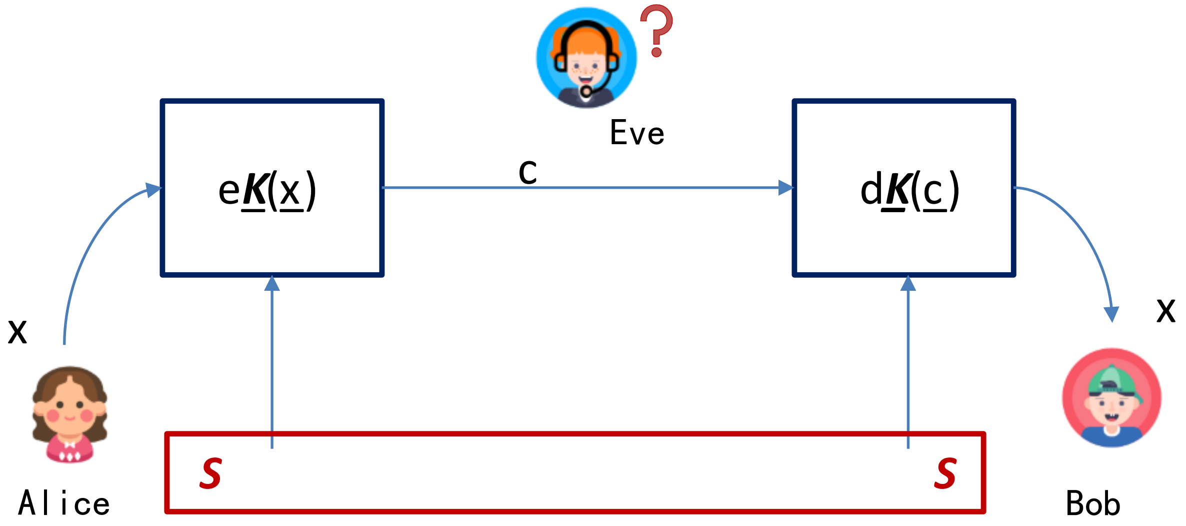
Bob

Known	Unknown
$p = 23$	
$g = 5$	
$b = 15$	a
$B = 5^{15} \bmod 23 = 19$	
$A = 8$	
$s = 8^{15} \bmod 23 = 2$	

- g = public (prime) base, known to Alice, Bob, and Eve. g = 5
- p = public (prime) modulus, known to Alice, Bob, and Eve. p = 23
- a = Alice's private key, known only to Alice. a = 6
- b = Bob's private key known only to Bob. b = 15
- A = Alice's public key, known to Alice, Bob, and Eve. A = ga mod p = 8
- B = Bob's public key, known to Alice, Bob, and Eve. B = gb mod p = 19

Public-key Cryptography

Secret-key vs. Public-key



Public-key Cryptography

Reference Material

- https://ee.stanford.edu/~hellman/publications/24.pdf
- Textbooks in Mathematics Cryptography Theory and Practice
- https://en.wikipedia.org/wiki/Diffie%E2%80%93Hellman_key_exchange

Cryptography

Summary

- Cryptology
  - ✓ Cryptography
    - ➢ Secret-key cryptosystems
      - ▪ Shannon’ s theory
      - ▪ Confusion and diffusion
      - ▪ Stream cipher and Block cipher
    - ➢ Public-key cryptosystems
      - ▪ Diffie and Hellman
      - ▪ Factoring Integers and Discrete Logarithm
      - ▪ RSA cryptosystems
      - ▪ Key distribution
  - ✓ Cryptanalysis

What Is Key Management System

What Is Key Management System?

Also Abbreviated As KMS

- KMS is a system designed to **manage cryptographic keys** that are used in encryption, decryption or hashing
- Successful key management is **critical** to the security of a system even if you have everything encrypted
- **Compromised keys** could cause your encrypted data to be exposed to the attackers
- The design of a KMS is often more **complex** than the data encryption itself.
- The design of KMS involves things like:
    - System policy
    - User training
    - User Authorization
    - Organization and department interactions
    - Outside interactions

What Does KMS Do?

Managing the Life Cycle of a Key

Birth → Death

Key Generation

- The creation of a crypto key
- Normally created by a **pseudorandom number generator**
- We want high "quality" of randomness

Key Storage

- How to secure the generated keys before storing on disk.
- Similar idea as TDE
- Normally performs a **wrap** before storing and a **unwrap** after reading from the disk

Key Renewal

- Concerns about the **validity** of the key before it needs a renewal
- If key is **expired**, it cannot be used unless renewed

Key Rotation

- **Regenerate** a new key

Key Revocation

- **Invalidate** a key
- Make a key unusable right away

Key Destruction

- **Remove** the key such that it no longer exists

Common KMS Terms

Common Terms

- KEK – Key Encryption Key
- Master Key – Another term for KEK
- DEK – Data Encryption Key
- MDEK – Master Data Encryption Key
- HKDF – Hash-based Key Derivation Function

Key Generation

Key Generation

The Birth of a Key

- Key generation is the process of generating keys in cryptography.
- A key is used to encrypt and decrypt data
- A device or a program used to generate keys is called a key generator or keygen.
- Keys can be symmetrical like AES or DES keys or asymmetrical like RSA
    - Symmetrical key algorithm uses a single shared key
    - Asymmetrical key algorithm uses a public and a private key
- Keys are generated using a Random Number Generator (PNG) or PseudoRandom Number Generator (PRNG).
- PRNG tends to be more random than PNG and Many application and encryption library utilize PRNG to generate random number to ensure the security.
- **More random = better and safer**

For Example: OpenSSL

- Uses PRNG as a basis to generate keys
- You can generate your own keys very easily with OpenSSL

- Generate 128-bit long symmetrical keys:

```
$ openssl rand 128 > mysecret.key
```

- Generate 4096-bit long RSA asymmetrical keys

```
$ openssl genrsa –out keyfile.key 4096
$ openssl rsa –in keyfile.key –pubout –out keyfile.pub
```

Private key

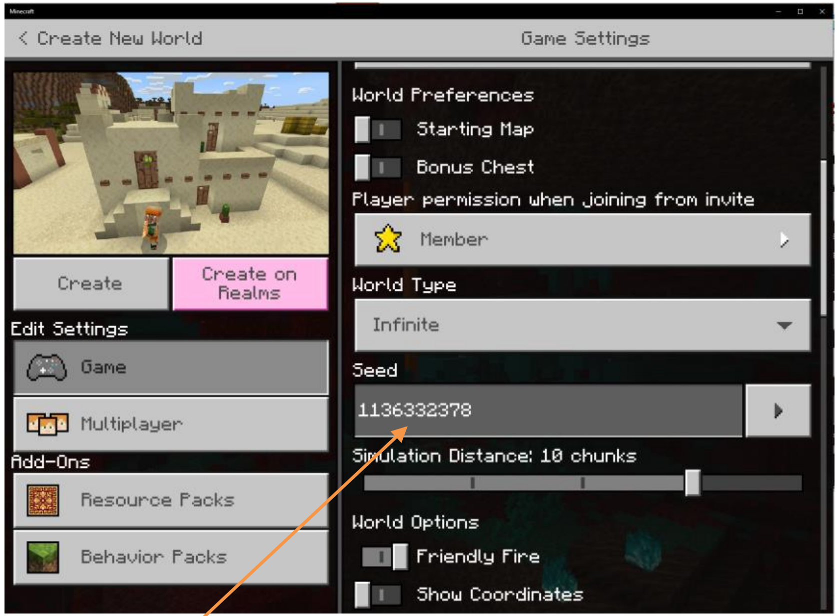
Public key

OpenSSL
Cryptography and SSL/TLS Toolkit

PseudoRandom Number Generator

The Birth of a Key

- PRNG uses system **entropy** to **seed** the data, which makes it hard to be guessed by attackers
- Entropy is the randomness collected by an operating system, often from hardware source such as fan or hard drive noise, mouse movement.
- Seed is a number used to initialize a PRNG
- Another way to generate randomness is to utilize information outside of the system such as a secret passphrase that you provide, or key derivation function (Hashing)
- Popular Simulation Game, **Minecraft**, uses PRNG in their world generation algorithm!



Create New World

Game Settings

World Preferences

Starting Map

Bonus Chest

Player permission when joining from invite

Member

World Type

Infinite

Seed

1136332378

Simulation Distance: 10 chunks

World Options

Friendly Fire

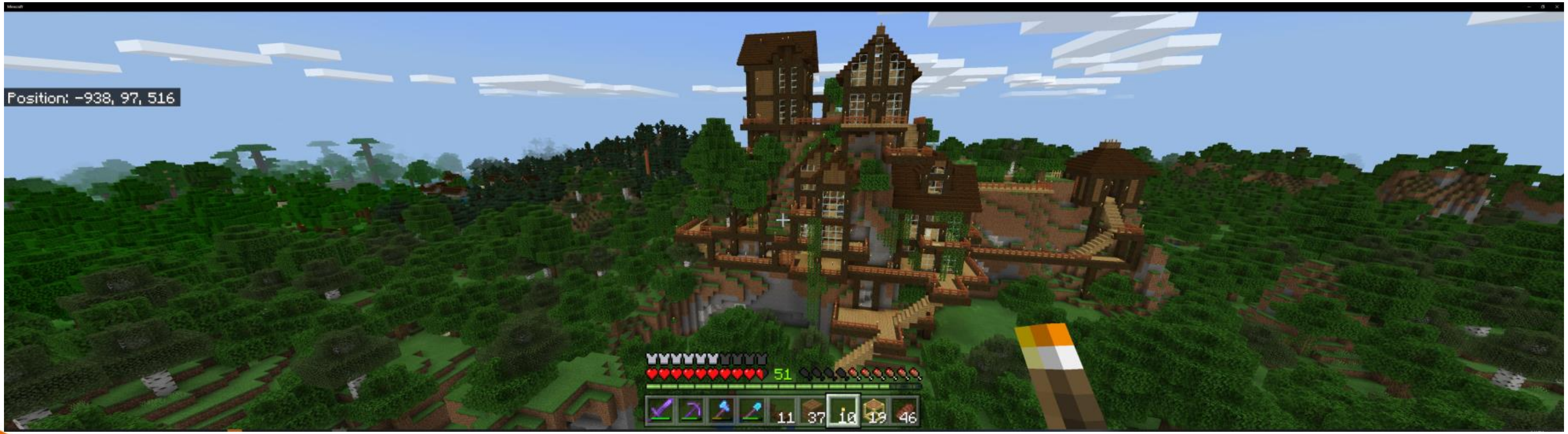
Show Coordinates

By Varying the Seed value, you could influence the world generation algorithm to start with a desert, mountain, forest, bamboo jungle terrain…etc

PseudoRandom Number Generator

The Birth of a Key

Something that I have built in Minecraft in the jungle…



Key Storage

Key Storage

Key Storage

- Now, we have generated the most secured key on the planet…
- How can we ensure KMS can store this key securely?
- This can be done in either 2-tier or 3-tier KMS.
- Both will involve a special key called Key Encryption Key (KEK). This key is sometimes referred as the Master Key.
- The KEK is used to wrap the encryption key before writing to storage
- And used to unwrap the wrapped encryption key after reading from storage.
- Wrap and unwrap are just other words for encrypt and decrypt when dealing with key materials, They are also a type of symmetrical encryption.

Question

What is KEK and where does it come from?

How to use KEK?

Key Storage

2 Tier Key Management

- 2 tier key management is a common KMS architecture to manage multiple data encryption keys (DEKs)
- Uses a password-based KEK generation
- KEK is the most important key component because it can basically be used to “**unlock**” all the DEKs that the system uses.
- **Protection** of KEK would be a primary focus on a 2-tier key management system architecture.

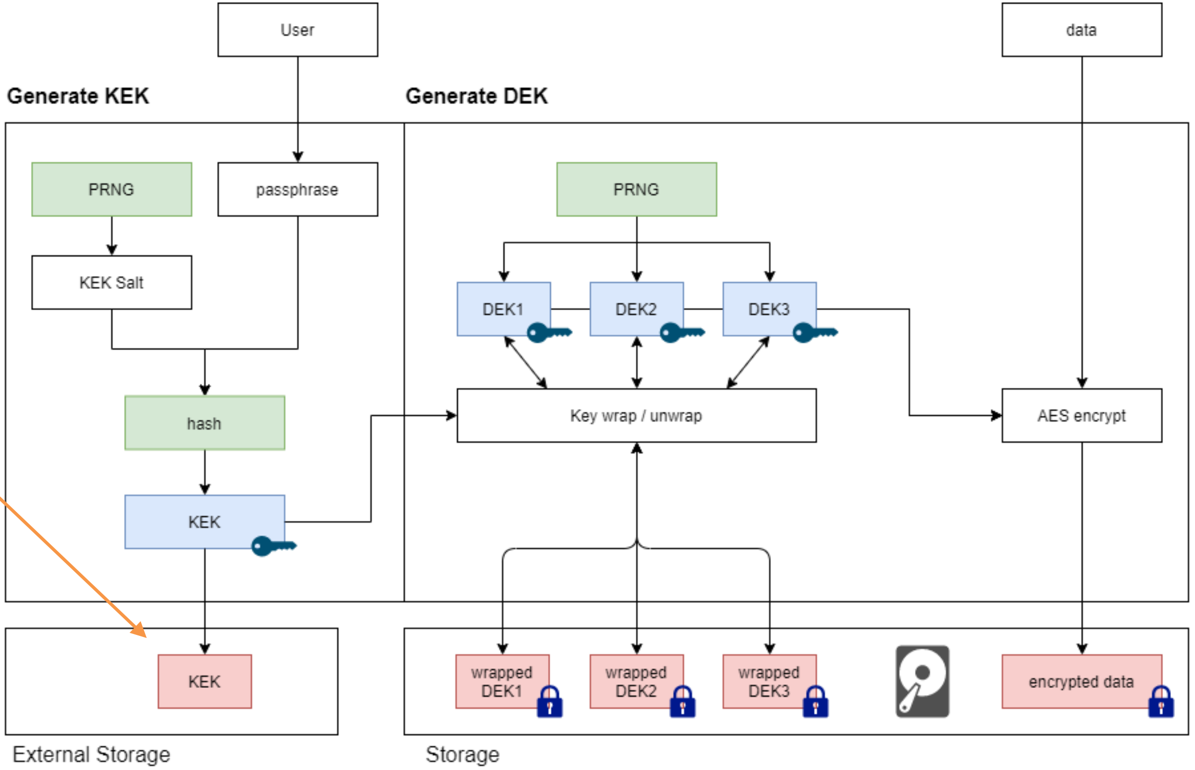
General Workflow

1. User enters a password
2. System generates a random number, perform calculation with user password and compute KEK
3. Store KEK in a secured location
4. System generate DEKs as random numbers
5. KEK wraps the DEKs and store them in a database server
6. DEKs are used to encrypt user data

Key Storage

2-tier Key Management

Saving KEK? Is it a Good Idea?



Key Storage

3 Tier Key Management

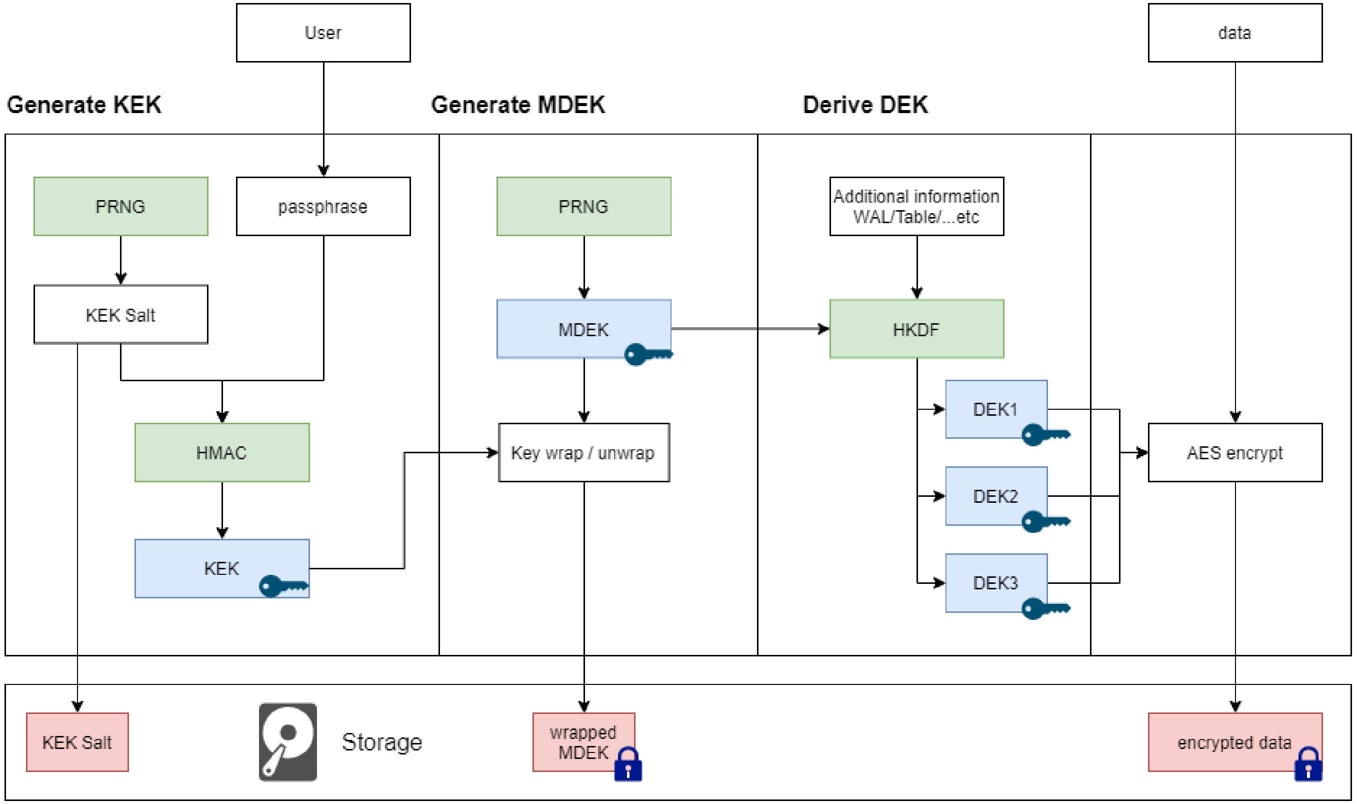
- 3 tier key management is a more complex KMS architecture to manage multiple data encryption keys (DEKs)
- KEK is also used to wrap other key material
- Less key materials are stored
- Uses Hash-based Key Derivation Function (**HKDF)** to derive the actual DEKs to be used

General Workflow

1. User enters a password
2. System generates a random number, perform calculation with user password and compute KEK
3. Store the random number used to compute KEK in a secured location
4. System generate MDEK as random number
5. Use MDEK plus additional information to derive each DEK
6. DEKs are used to encrypt user data

Key Storage

3-tier Key Management



PEKING UNIVERSITY

HIGH GO

Key Renewal

Key Renewal

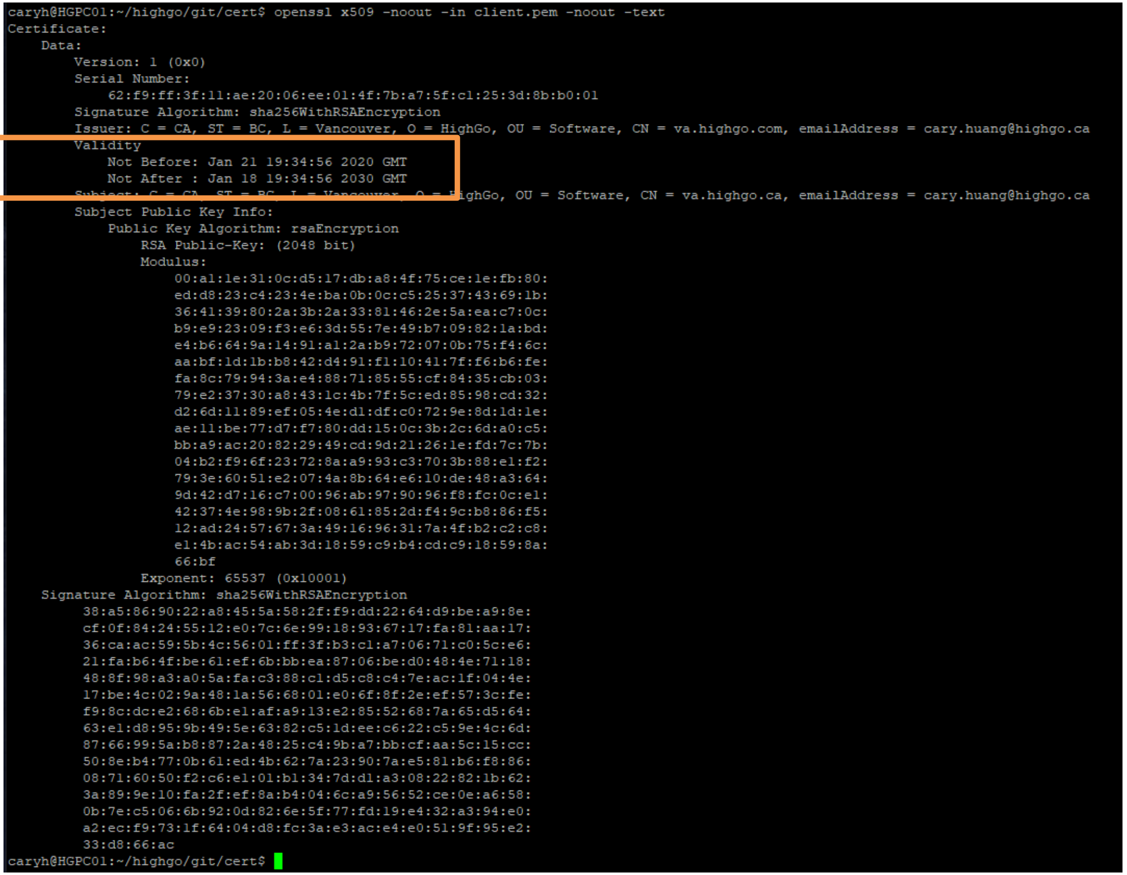
Validity of Key Materials

- The protection of a key material is important.
- Knowing how long a key can be used for is also very important.
- An expired key is considered unsafe in today’s standard.
- The concept of key validity is most often used in an X509 certificate where both client and server use asymmetrical encryption (**public** and **private** keys) to mutually authenticate each other.

Key Renewal

X509 Certificate Validity

- When we examine the details of a X509 certificate, we can see the value of the public key and also the **validity periods.**
- Represented by “**Not Before**” and “**Not After**” parameters
- Using the X509 certificate before the validity period begins is also considered invalid!



```
caryh@HGPC01:~/highgo/git/cert$ openssl x509 -noout -in client.pem -noout -text
Certificate:
    Data:
        Version: 1 (0x0)
        Serial Number:
            62:f9:ff:3f:11:ae:20:06:ee:01:4f:7b:a7:5f:c1:25:3d:8b:b0:01
        Signature Algorithm: sha256WithRSAEncryption
        Issuer: C = CA, ST = BC, L = Vancouver, O = HighGo, OU = Software, CN = va.highgo.com, emailAddress = cary.huang@highgo.ca
        Validity
            Not Before: Jan 21 19:34:56 2020 GMT
            Not After : Jan 18 19:34:56 2030 GMT
        Subject: C = CA, ST = BC, L = Vancouver, O = HighGo, OU = Software, CN = va.highgo.ca, emailAddress = cary.huang@highgo.ca
        Subject Public Key Info:
            Public Key Algorithm: rsaEncryption
                RSA Public-Key: (2048 bit)
                Modulus:
                    00:a1:1e:31:0c:d5:17:db:a8:4f:75:ce:1e:fb:80:
                    ed:d8:23:c4:23:4e:ba:0b:0c:c5:25:37:43:69:1b:
                    36:41:39:80:2a:3b:2a:33:81:46:2e:5a:ea:c7:0c:
                    b9:e9:23:09:f3:e6:3d:55:7e:49:b7:09:82:1a:bd:
                    e4:b6:64:9a:14:91:a1:2a:b9:72:07:0b:75:f4:6c:
                    aa:bf:1d:1b:b8:42:d4:91:f1:10:41:7f:f6:b6:fe:
                    fa:8c:79:94:3a:e4:88:71:85:55:cf:84:35:cb:03:
                    79:e2:37:30:a8:43:1c:4b:7f:5c:ed:85:98:cd:32:
                    d2:6d:11:89:ef:05:4e:d1:df:c0:72:9e:8d:1d:1e:
                    ae:11:be:77:d7:f7:80:dd:15:0c:3b:2c:6d:a0:c5:
                    bb:a9:ac:20:82:29:49:cd:9d:21:26:1e:fd:7c:7b:
                    04:b2:f9:6f:23:72:8a:a9:93:c3:70:3b:88:e1:f2:
                    79:3e:60:51:e2:07:4a:8b:64:e6:10:de:48:a3:64:
                    9d:42:d7:16:c7:00:96:ab:97:90:96:f8:fc:0c:e1:
                    42:37:4e:98:9b:2f:08:61:85:2d:f4:9c:b8:86:f5:
                    12:ad:24:57:67:3a:49:16:96:31:7a:4f:b2:c2:c8:
                    e1:4b:ac:54:ab:3d:18:59:c9:b4:cd:c9:18:59:8a:
                    66:bf
                Exponent: 65537 (0x10001)
    Signature Algorithm: sha256WithRSAEncryption
         38:a5:86:90:22:a8:45:5a:58:2f:f9:dd:22:64:d9:be:a9:8e:
         cf:0f:84:24:55:12:e0:7c:6e:99:18:93:67:17:fa:81:aa:17:
         36:ca:ac:59:5b:4c:56:01:ff:3f:b3:c1:a7:06:71:c0:5c:e6:
         21:fa:b6:4f:be:61:ef:6b:bb:ea:87:06:be:d0:48:4e:71:18:
         48:8f:98:a3:a0:5a:fa:c3:88:c1:d5:c8:c4:7e:ac:1f:04:4e:
         17:be:4c:02:9a:48:1a:56:68:01:e0:6f:8f:2e:ef:57:3c:fe:
         f9:8c:dc:e2:68:6b:e1:af:a9:13:e2:85:52:68:7a:65:d5:64:
         63:e1:d8:95:9b:49:5e:63:82:c5:1d:ee:c6:22:c5:9e:4c:6d:
         87:66:99:5a:b8:87:2a:48:25:c4:9b:a7:bb:cf:aa:5c:15:cc:
         50:8e:b4:77:0b:61:ed:4b:62:7a:23:90:7a:e5:81:b6:f8:86:
         08:71:60:50:f2:c6:e1:01:b1:34:7d:d1:a3:08:22:82:1b:62:
         3a:89:9e:10:fa:2f:ef:8a:b4:04:6c:a9:56:52:ce:0e:a6:58:
         0b:7e:c5:06:6b:92:0d:82:6e:5f:77:fd:19:e4:32:a3:94:e0:
         a2:ec:f9:73:1f:64:04:d8:fc:3a:e3:ac:e4:e0:51:9f:95:e2:
         33:d8:66:ac
caryh@HGPC01:~/highgo/git/cert$
```

Key Renewal

X509 Certificate Validity

- How to **renew** the validity period and maintaining the **same** set of public and private keys in a X509 certificate?
- Again you can use Openssl command line tool
- It will take several steps to renew the validity period within a certificate without generating a new one.
- If you are interested in this process, you can refer to this link right here:

https://uwnthesis.wordpress.com/2019/06/13/openssl-how-to-generate-a-2nd-certificate-csr-from-an-existing-private-key-so-that-the-new-cert-contains-the-same-public-key-as-the-expired-cert/

Key Renewal

How About the Validity of Symmetrical Keys

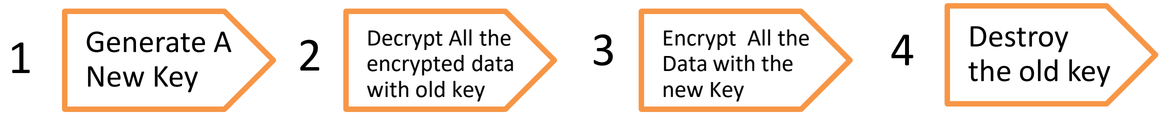
- For symmetrical keys, there is not a concept of validity by themselves.
- You generate an symmetrical key with OpenSSL, and you can use it forever and no one can stop you from doing that…
- When the symmetrical key is managed by a KMS, that is a different story…
- This is up to the KMS to provide an extra layer of security (such as validity) on top of the symmetrical keys that it is managing.
- **For example**, if an application requests a KEK from a KMS, it could refuse this operation because the KEK is expired and escalate to a key administrator to resolve this.

Key Rotation

Key Rotation

What is Key Rotation?

- Key rotation simply means to change a key. It could be DEK, or KEK… etc.
- Key rotation is normally applied to symmetrical keys because of its simplicity.
- Key rotation is more complicated on asymmetrical keys because instead of 1 key, it is a pair of keys and both are related. Some KMS simply does not support this.
- The process normally goes like this:


1
Generate A New Key
2
Decrypt All the encrypted data with old key
3
Encrypt All the Data with the new Key
4
Destroy the old key

Key Rotation

Why Rotate Keys

- Periodically and automatically rotating keys is a recommended security practice.
- Industry standards and compliances, such as Payment Card Industry Data Security Standard (PCI DSS), require the regular rotation of keys.
- If a key is compromised, regular rotation limits the number of actual messages vulnerable to compromise.
- Regular key rotation ensures that your system is resilient to manual rotation.
- Generally, regular key rotation makes a system safer.

Key Rotation

How often to rotate?

- Automatic and periodic key rotation is generally recommended.
- The rotation schedule can be defined based on a key's age or on the volume of data encrypted.
- Again, it is dependent on the security compliance in some industry.
- Some requires a rotation at a fixed period like every 90 days.
- You should also manually rotate the key if you suspect it has been compromised.

- Rotate every day?
- Rotate every week?
- Rotate every month?
- Rotate every 3 months?
- Rotate every year?
- Rotate every 64 GB?
- Rotate every 128GB?

Key Rotation

A question to think about

1 Generate A New Key

2 Decrypt All the encrypted data with old key

3 Encrypt All the Data with the new Key

4 Destroy the old key

Got a new key here

Hmm… but my database size is 2TB… So when I rotate a key, I have to:

- Load data from disk to memory
- Decrypt that data with old key
- Encrypt that data with new key
- Write back to disk
- Load another chunk of data from disk and repeat

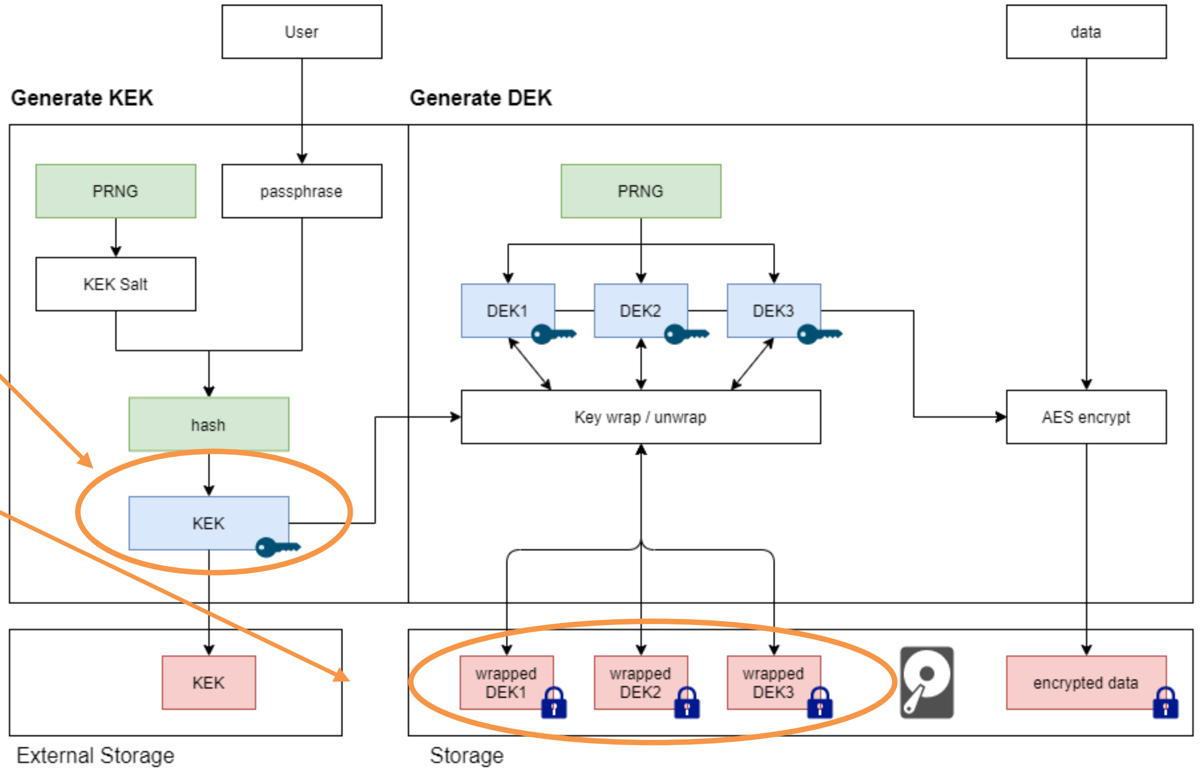
That sounds like a boring process that will take forever to complete.

What should we do instead?

Key Rotation

How to Rotate Efficiently?

- Simple! Instead of rotating the DEKs that encrypt the actual data, we can rotate the KEK.
- We Rotate this guy here
- Then all we have to do is unwrap the DEKs with old key, wrap with the new key and write back to disk.
- This should be done very quickly

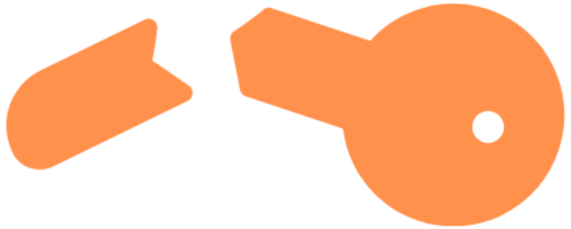


Key Revocation and Destruction

Key Revocation and Destruction

The end of life

- When a key has served its purpose, it is time to retire.
- You can either "**Revoke**" a key or "**Destroy**" a key.
- Revoke means to "**disable**" a key, meaning the key can no longer be used to do encryption.
- Revoked key can be "**re-enabled**" to be used again
- A "**destroyed**" key is gone forever, so you really need to pay close attention here before destroying a key.
- Make sure no data is encrypted with the key that is about to get destroyed.
- Otherwise, you will end up with "**stranded**" data. (Data that no one is able to make use of).

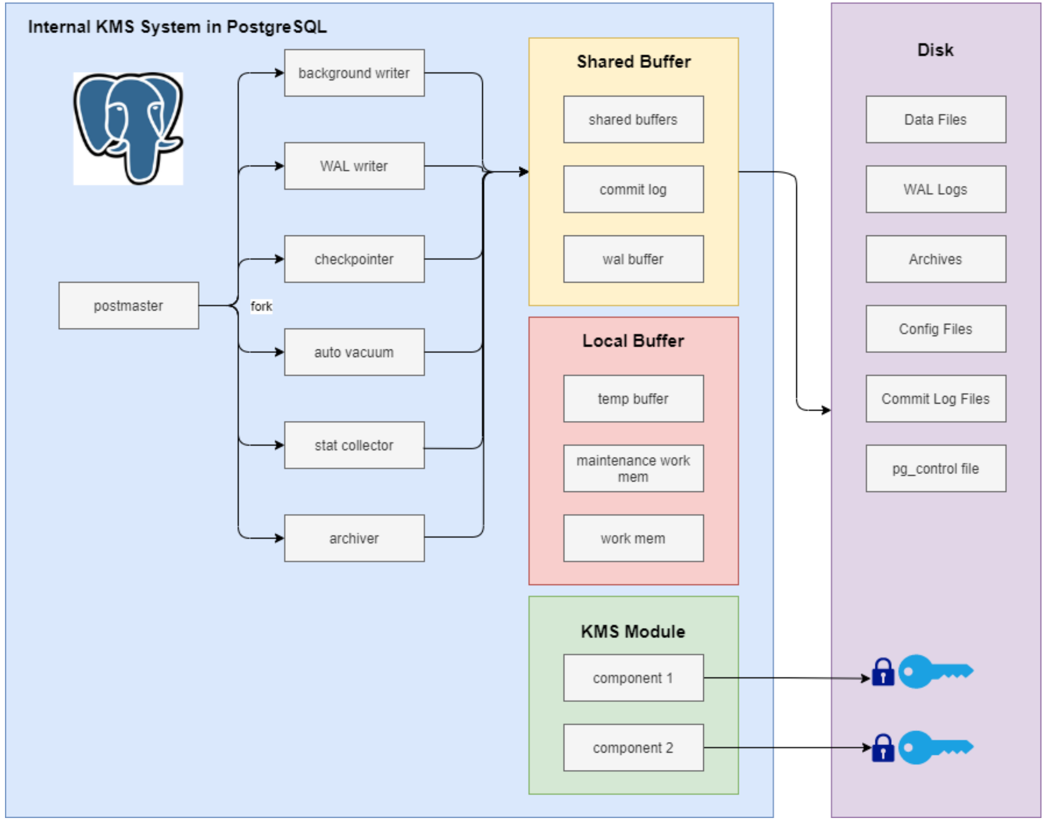


External KMS vs Internal KMS

External KMS vs Internal KMS

Internal KMS

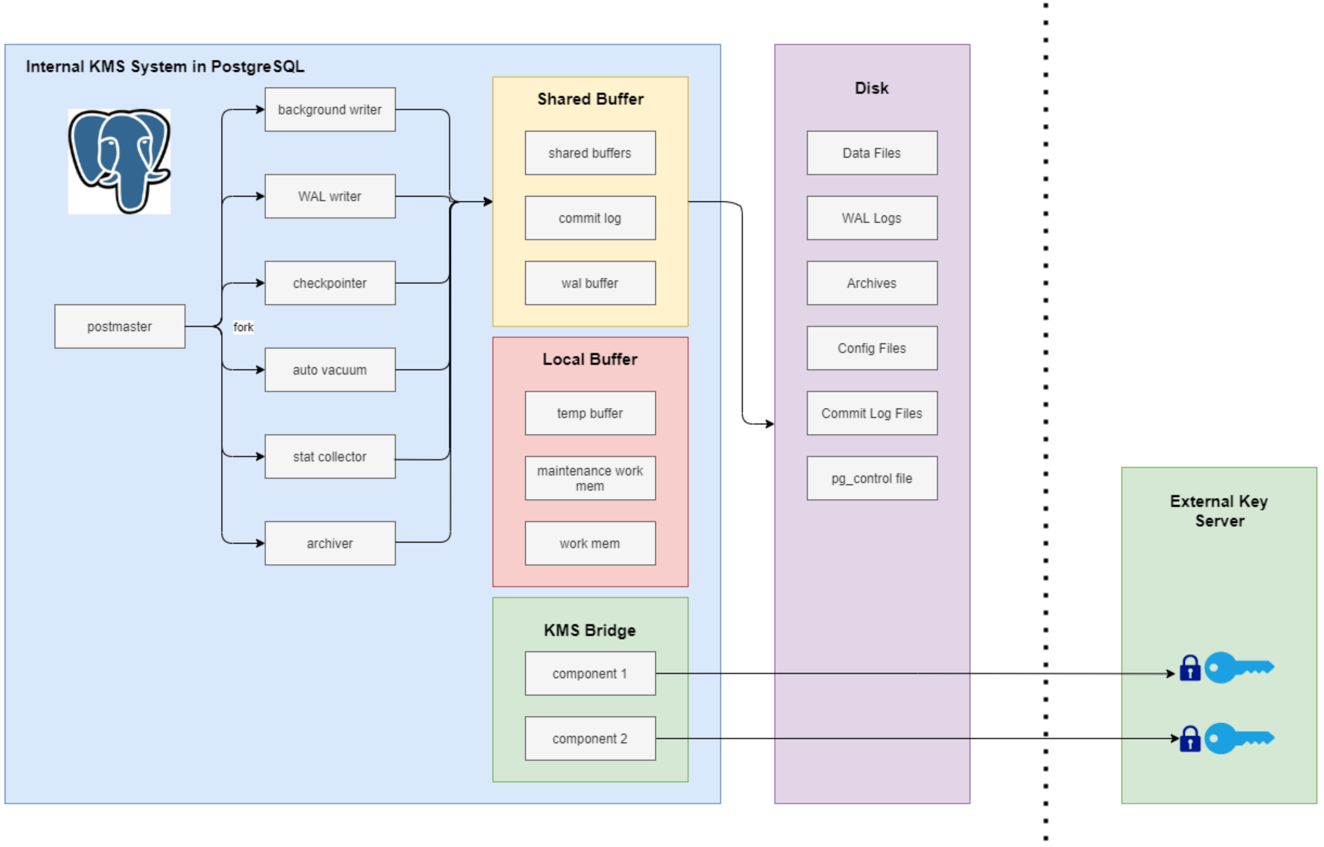
- Internal KMS means the KMS is built within the application as a separate module or component
- The keys are stored in the same disk space as the application



External KMS vs Internal KMS

External KMS

- External KMS means the KMS is built outside of the application
- The keys are stored in the separately from the application
- Normally there is a KMS Bridge component build in the application to communicate with external KMS



Benefits of External KMS

Compliance and Security Audit

Better Compliance	Easier Security Audit
• For organizations that are mandated by the government to have the highest security compliance certification • For example, in the Payment Card Industry Data Security Standard (PCI DSS) requires that the cardholder data and encryption keys must be protected and stored separately	• Required by certain corporate and industry compliance. • Compliance requires detailed audit log capturing all the key usages, rotations, who accessed the key and at what time. • With an external KMS system, it tends to be much easier to streamline the key audit reports for all the keys it is managing. • Easier to prove to customers or potential auditors that the keys are indeed very secured and closely monitored.

Benefits of External KMS

Separation of Duty and Scalability

Separation of Duty	Scalability
• The key administrators of an external KMS has the ability to configure permissions for all the cryptographic keys that it manages. • Permissions such as intended purpose, owner, validity period and other user attributes. • Internal KMS may not have this level of granularity. • For PostgreSQL, a database super user could also be the key administrator. • This may be an issue with some compliance that both roles must be separated	• As system grows bigger and more complex, the number of keys are expected to grow as well. • Each key has a series of attributes that should be managed. • External KMS is normally designed to be very scalable as the application grows. • Internal KMS, however, is normally not designed to scale and as system grows, the application needs to migrate all the keys to an external KMS.

瀚高
HIGH GO

融知与行 瀚且高远

THANKS