Vesper

Vector Embedded Super-Positioning Encryption Rules

Real-World Use & Structure Focused

Symmetric? Asymmetric?

Why not take advantages of both?

AES vs. RSA: The Two Major Players

- Even since God created the world, there have been many cryptic texts for humans to decipher.
  - As of today (21st century), there are two major parties called *symmetric* and *asymmetric*.
  - The war between two parties continues, and their comparison summary is here:

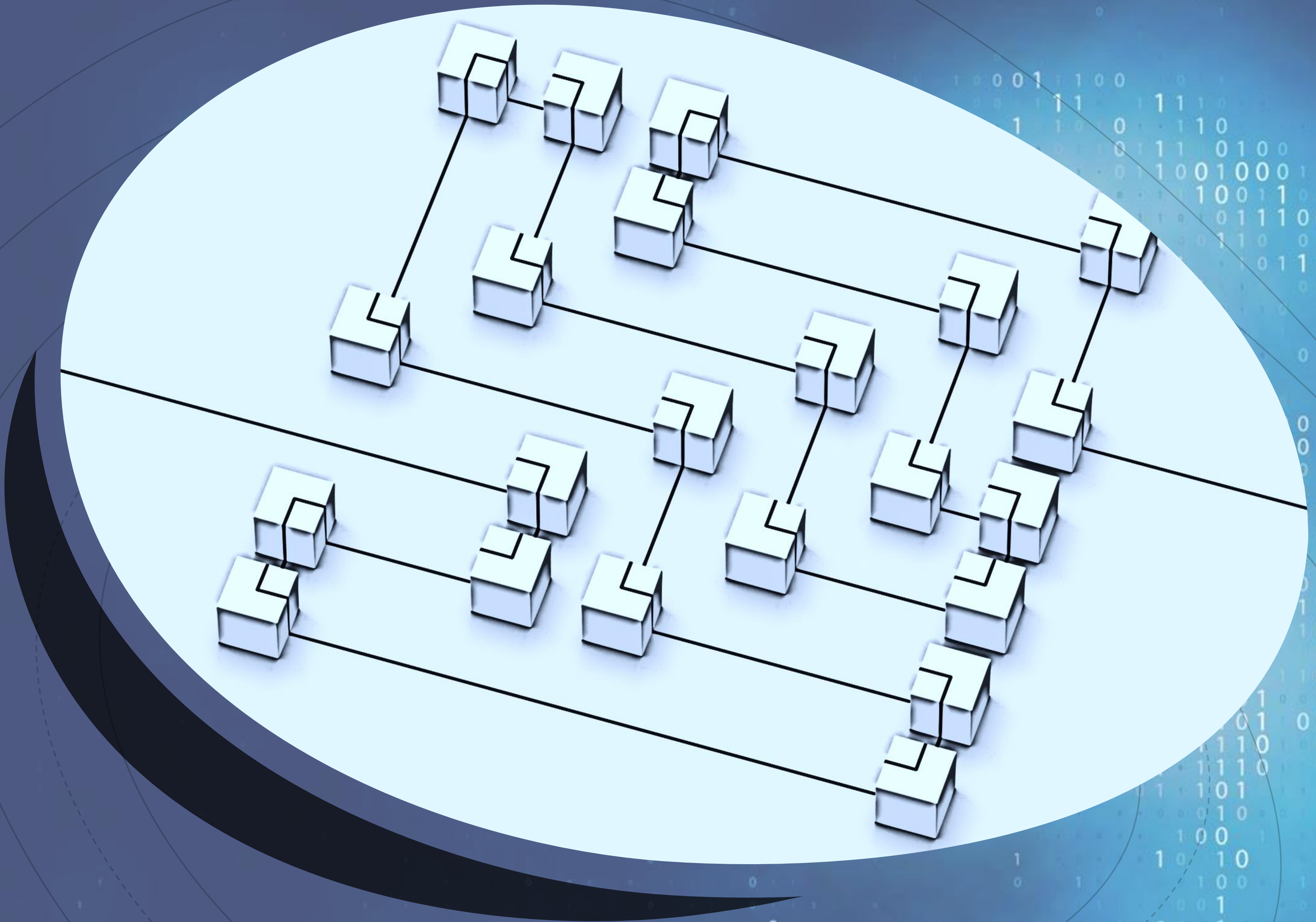
    (source: International Journal of Scientific Research in Computer Science, Engineering and Information Technology, Vol. 3, Issue 4, ISSN: 2456-3307)

	AES	RSA
Approach	Symmetric	Asymmetric
Encryption	Fast	Slow
Decryption	Fast	Slow
Key Distribution	Difficult	Easy
Complexity	O($\log N$)	O(N^3)
Security	Moderate	Highest
Nature	Closed	Open

- The debate continues because of:
  - Their distinctive characteristics: slightly different uses
  - Different resource (storage, performance, etc.) requirements
  - Managerial issue: especially key management (storage and transfer)
  - Government intervention: (no comment on this one)
  - Commercial market trend: (also no comment on this one… oh, hello Intel®)
- Q: Is there any way to take advantages of (from) both sides?
  - A: Yes, there is.
- Q: If 'yes', then does user have to do something or learn (yuk!) something?
  - A: No, not at all.
- Q: Then, what is the answer?
  - A: The answer is *Vesper*, a vector-based encryption algorithm with the revolutionary new feature called 'Key-morphing'.

What is Vesper?

- General Feature
  - Vector-based cipher: using moving vectors on lattice/grid
    - Totally different concept of a ‘vector’ as used by the block ciphers
  - Symmetric key cipher
  - Not a block cipher – processes data ‘byte-by-byte’
  - Uses 1024-byte (8192-bit) key (from version 1.3 and later)
  - Cipher text size is twice (yes, ‘×2’) as big as plain text (for prototype version)
  - Super fast: enough to handle (encrypt/decrypt) 4K HD data in real-time
    - Fast enough to replace stream ciphers
- Special feature (that no other common ciphers can possibly provide)
  - Search/match data (including texts) using cipher text (no decryption, no artifact)
  - Perform calculation/evaluation using cipher text (no decryption, no artifact)
    - Basic arithmetic (+, –, ×, ÷) and evaluation/validation (>, ≥, =, ≤, <) operations
  - Key-morphing
    - Convert cipher text from key A-based to key B-based without decryption (no artifact)

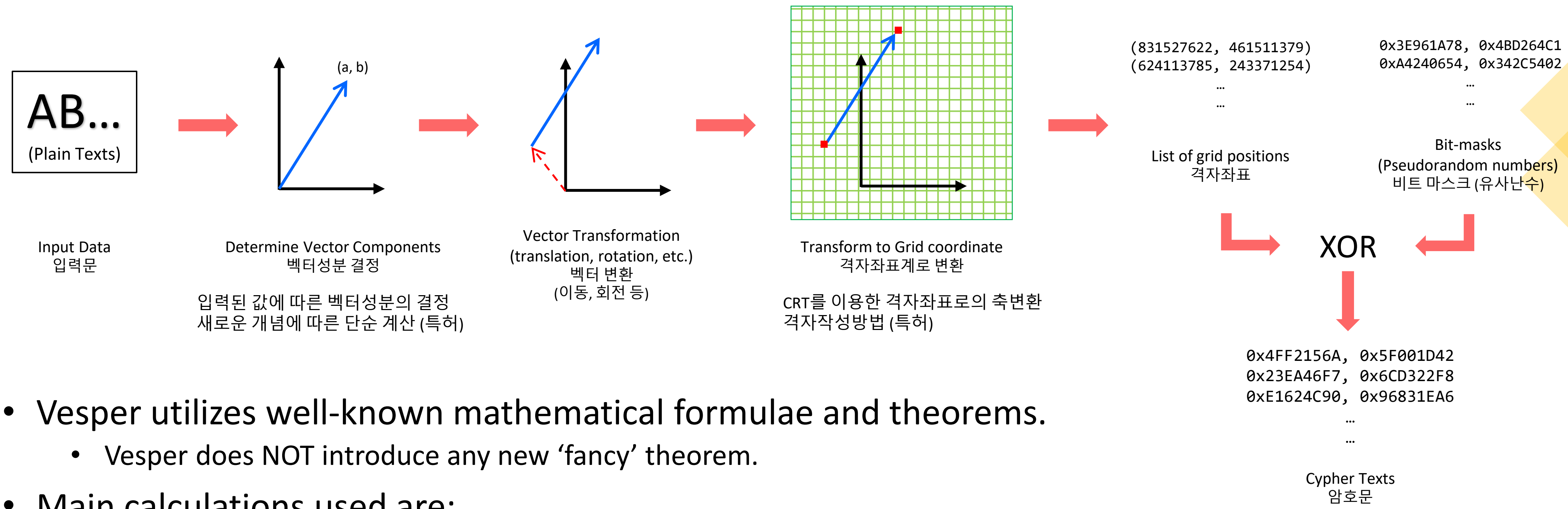


Vesper Encryption Process

Underlying

Logics & Mathematics

Encryption Process



- Vesper utilizes well-known mathematical formulae and theorems.
  - Vesper does NOT introduce any new ‘fancy’ theorem.
- Main calculations used are:
  - Linear equation (1차 방정식): $ax + by + c = 0$
  - Vector transformation (벡터변환): Translation (평행이동), Rotation (회전이동), etc.
  - Chinese remainder theorem (CRT): 중국인의 나머지 정리
  - XOR (exclusive-OR) bit operation (배타적 논리합)
  - Pseudorandom number generation (similar to the OTP bank key): 유사난수

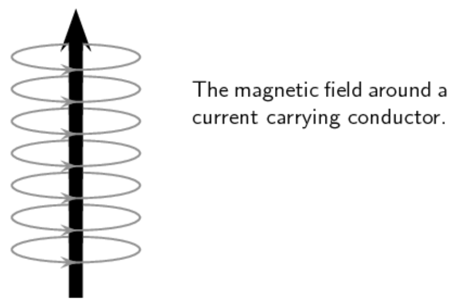
Search

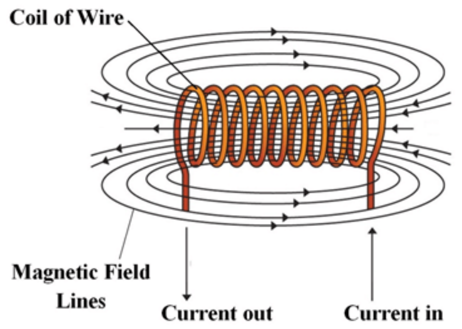
&

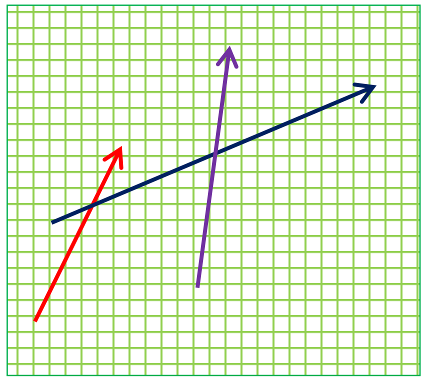
Replace

Sure, it was considered impossible because Vesper wasn't invented yet.

Vector Embedded, Super-Positioned Plain Text

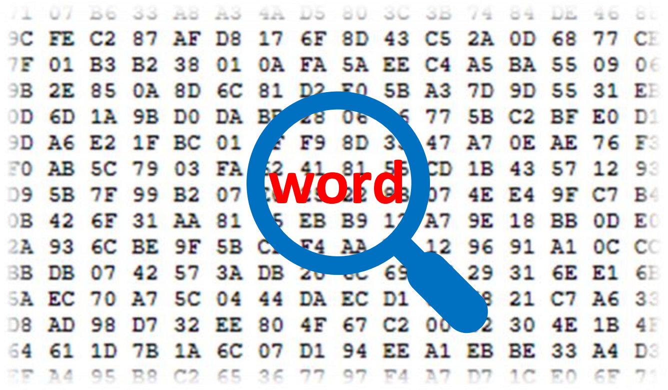


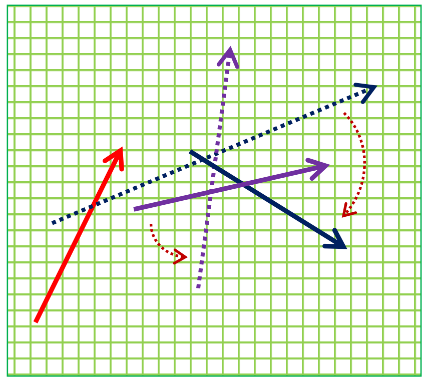




- Vectors in the Vesper universe contain more than just numbers.
  - Existing cryptos manipulate plain texts to hide the real values; Vesper doesn't.
- Vesper algorithm 'transform' the plain texts to create vectors that exist in the Vesper grid.
  - Plain texts exist with cipher texts simultaneously but in different '*super-positioned*' space.
    - Just like the magnetic field exists along the path through where electric currents flows.
- The grid (lattice) coordinate system that Vesper employs efficiently hides the plain text.
  - Vector analysis can reveal the plain texts in the Vesper grid system.

Search, then replace



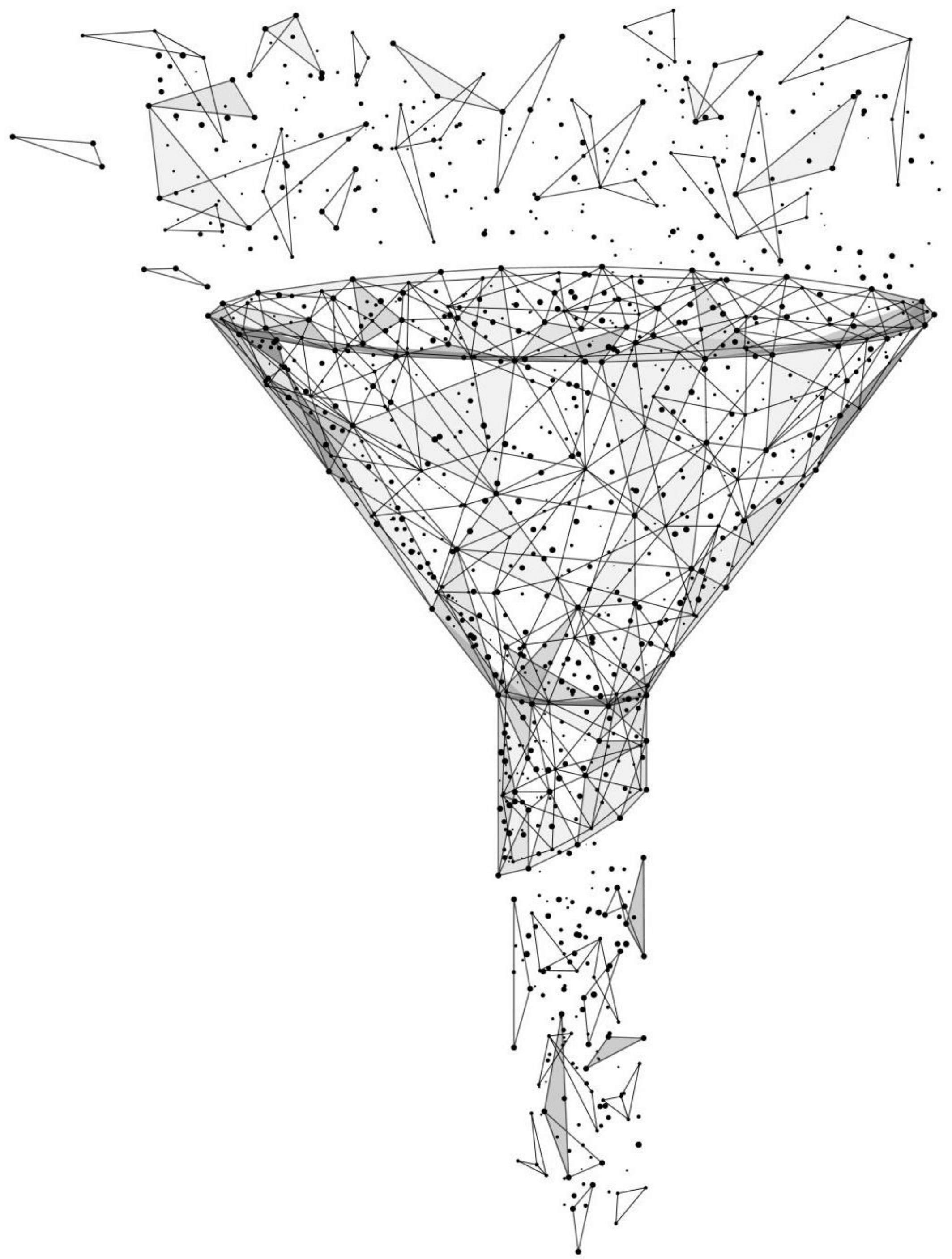


- For Vesper, searching cipher text for specific data is not much different than doing so in plain text.
  - For vesper's simple – and fast – calculation, plain text is clearly visible within the Vesper lattice.
- Vesper's byte search algorithm is different from normal search algorithm.
  - It was designed for a different purpose in 90's by Si Park. (A new patent will be applied for.)
- Replace is just a simple vector transformation.
  - Once the target data is found, recalculating the vector components will translate the vector → replacement.
  - 'Original' and 'new' data must have the same byte-length, of course.

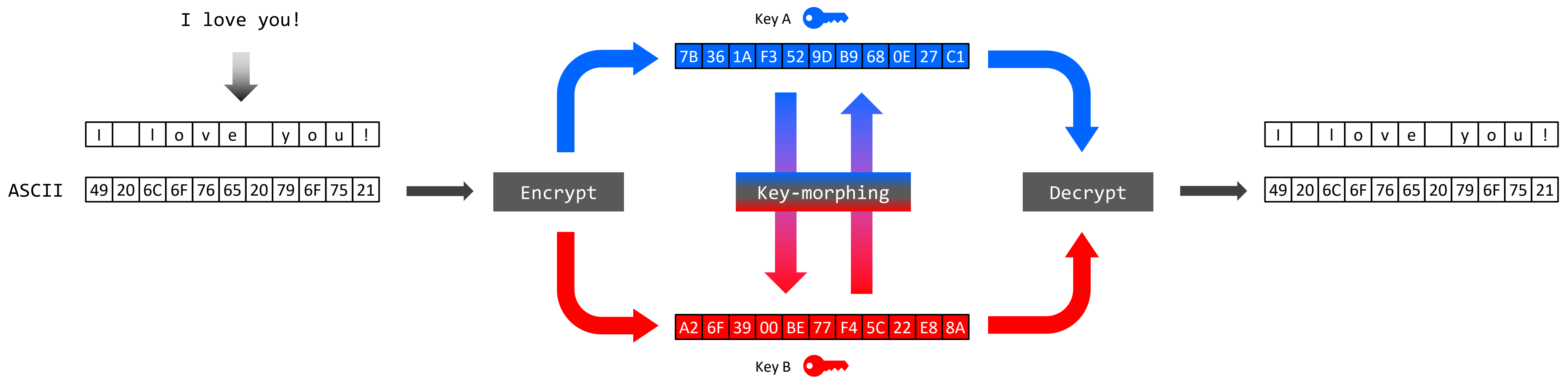
Instantaneous Encrypted Data Morphing

Changing Safe Lock Combination

Without Opening the Safe



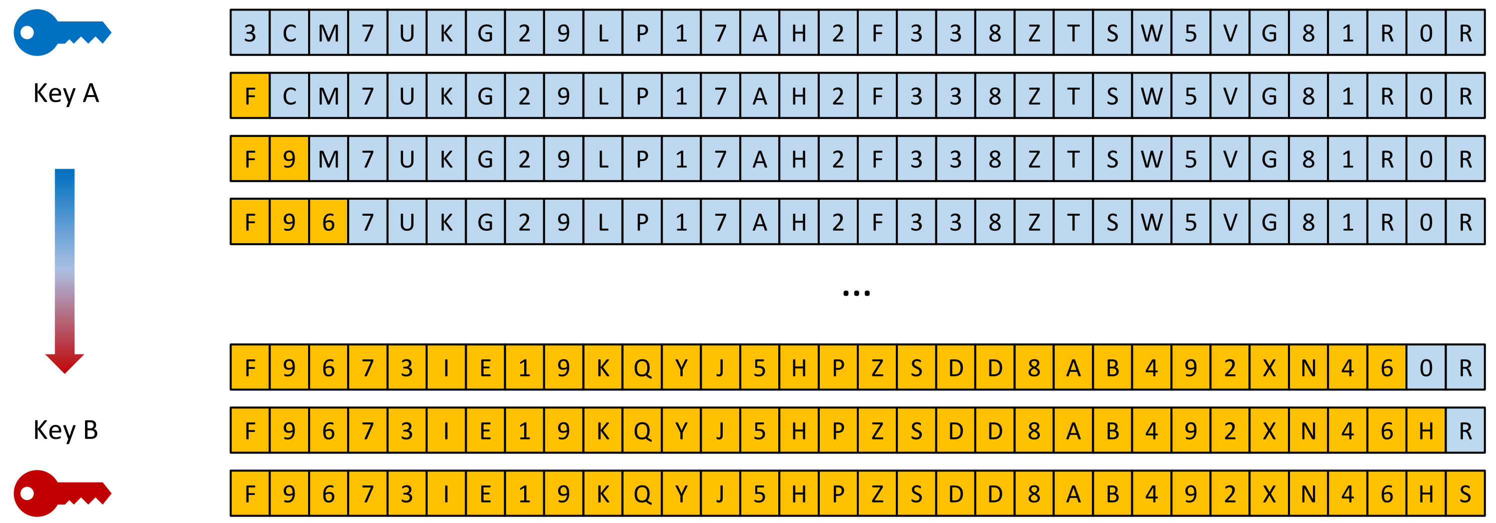
Key-morphing Overview



I love you!
I | | l | o | v | e | | y | o | u | !
ASCII
49 | 20 | 6C | 6F | 76 | 65 | 20 | 79 | 6F | 75 | 21
Encrypt
Key A
7B 36 1A F3 52 9D B9 68 0E 27 C1
Key-morphing
A2 6F 39 00 BE 77 F4 5C 22 E8 8A
Key B
Decrypt
I | | l | o | v | e | | y | o | u | !
49 | 20 | 6C | 6F | 76 | 65 | 20 | 79 | 6F | 75 | 21

- Key-morphing converts cipher text ‘A’ to ‘B’ without decryption process.
- No data is saved and/or stored during the key-morphing process.
  - **No artifact** left either on the disk or in the memory.
- Internally, it’s done by vector calculation.
  - Simple calculation to move around the vectors that represent cipher text – very *fast*!

Morphing? Like ‘Mystique’ from X-Men?



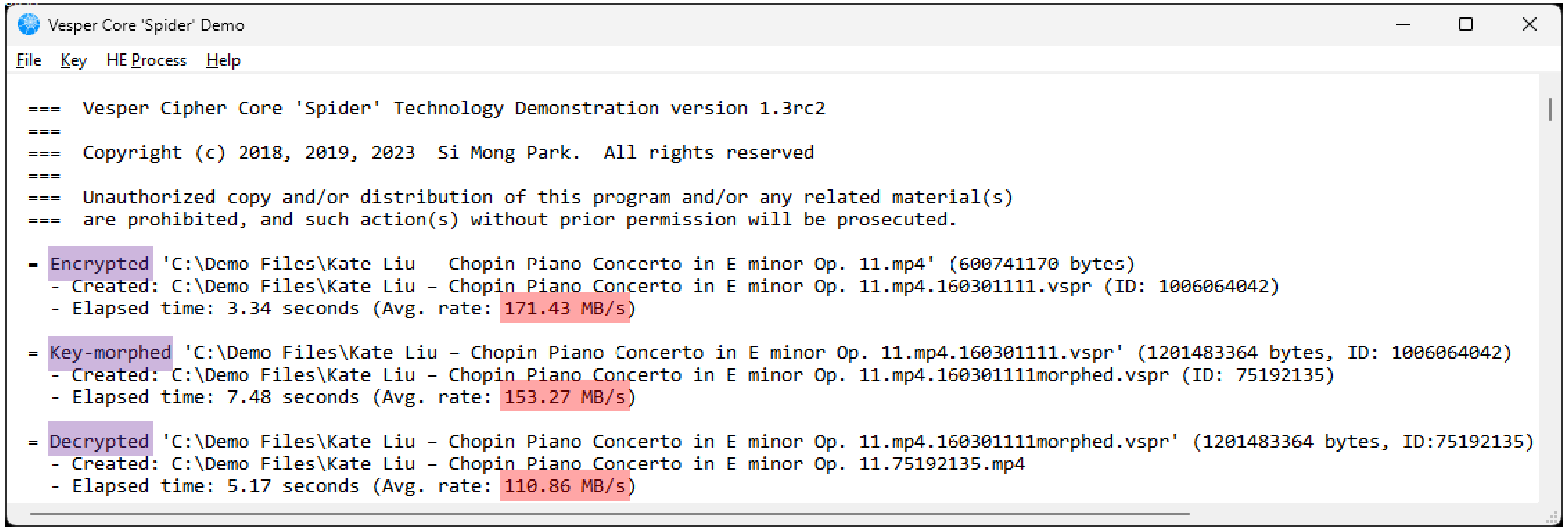
3	C	M	7	U	K	G	2	9	L	P	1	7	A	H	2	F	3	3	8	Z	T	S	W	5	V	G	8	1	R	0	R
F	C	M	7	U	K	G	2	9	L	P	1	7	A	H	2	F	3	3	8	Z	T	S	W	5	V	G	8	1	R	0	R
F	9	M	7	U	K	G	2	9	L	P	1	7	A	H	2	F	3	3	8	Z	T	S	W	5	V	G	8	1	R	0	R
F	9	6	7	U	K	G	2	9	L	P	1	7	A	H	2	F	3	3	8	Z	T	S	W	5	V	G	8	1	R	0	R

…

F	9	6	7	3	I	E	1	9	K	Q	Y	J	5	H	P	Z	S	D	D	8	A	B	4	9	2	X	N	4	6	0	R
F	9	6	7	3	I	E	1	9	K	Q	Y	J	5	H	P	Z	S	D	D	8	A	B	4	9	2	X	N	4	6	H	R
F	9	6	7	3	I	E	1	9	K	Q	Y	J	5	H	P	Z	S	D	D	8	A	B	4	9	2	X	N	4	6	H	S

- Actually, *YES!*
- Morphing occurs ‘byte-by-byte’ within the CPU without storing or saving the intermediate data.
  - No artifact.
- Instantly invalidates ‘old’ key.
- It is *faster* than decrypting and re-encrypting entire data.
  - It also uses less disk space because the plain text data is NOT generated.

So, how *fast* is the key-morphing process?

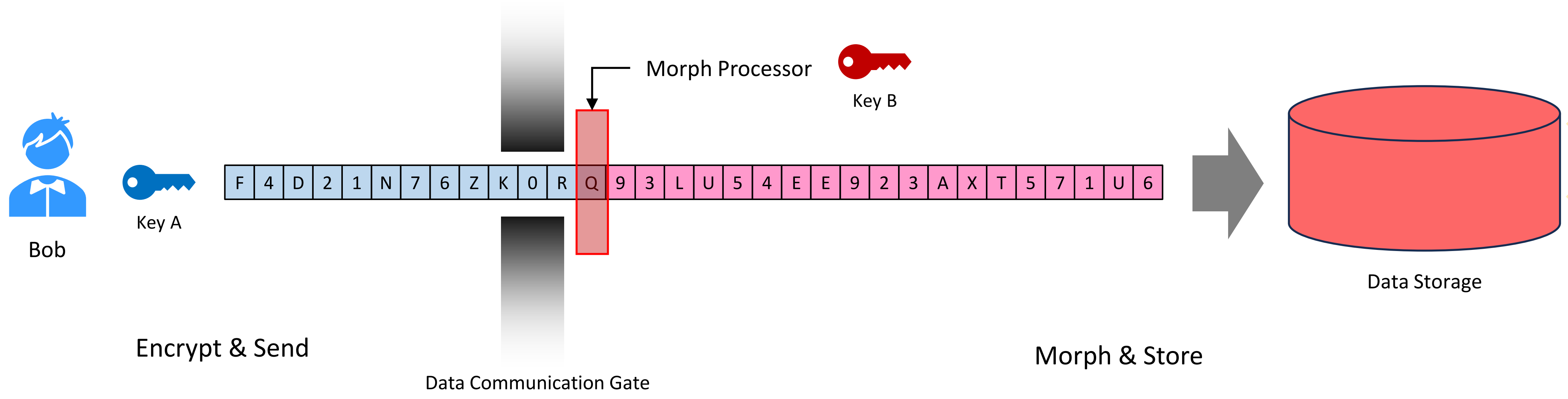


```
===  Vesper Cipher Core 'Spider' Technology Demonstration version 1.3rc2
===
===  Copyright (c) 2018, 2019, 2023  Si Mong Park.  All rights reserved
===
===  Unauthorized copy and/or distribution of this program and/or any related material(s)
===  are prohibited, and such action(s) without prior permission will be prosecuted.

= Encrypted 'C:\Demo Files\Kate Liu - Chopin Piano Concerto in E minor Op. 11.mp4' (600741170 bytes)
  - Created: C:\Demo Files\Kate Liu - Chopin Piano Concerto in E minor Op. 11.mp4.160301111.vspr (ID: 1006064042)
  - Elapsed time: 3.34 seconds (Avg. rate: 171.43 MB/s)

= Key-morphed 'C:\Demo Files\Kate Liu - Chopin Piano Concerto in E minor Op. 11.mp4.160301111.vspr' (1201483364 bytes, ID: 1006064042)
  - Created: C:\Demo Files\Kate Liu - Chopin Piano Concerto in E minor Op. 11.mp4.160301111morphed.vspr (ID: 75192135)
  - Elapsed time: 7.48 seconds (Avg. rate: 153.27 MB/s)

= Decrypted 'C:\Demo Files\Kate Liu - Chopin Piano Concerto in E minor Op. 11.mp4.160301111morphed.vspr' (1201483364 bytes, ID:75192135)
  - Created: C:\Demo Files\Kate Liu - Chopin Piano Concerto in E minor Op. 11.75192135.mp4
  - Elapsed time: 5.17 seconds (Avg. rate: 110.86 MB/s)
```

(Time measured with the 572MB data file under desktop PC under Windows 11 Pro with i9-12900KF CPU.)

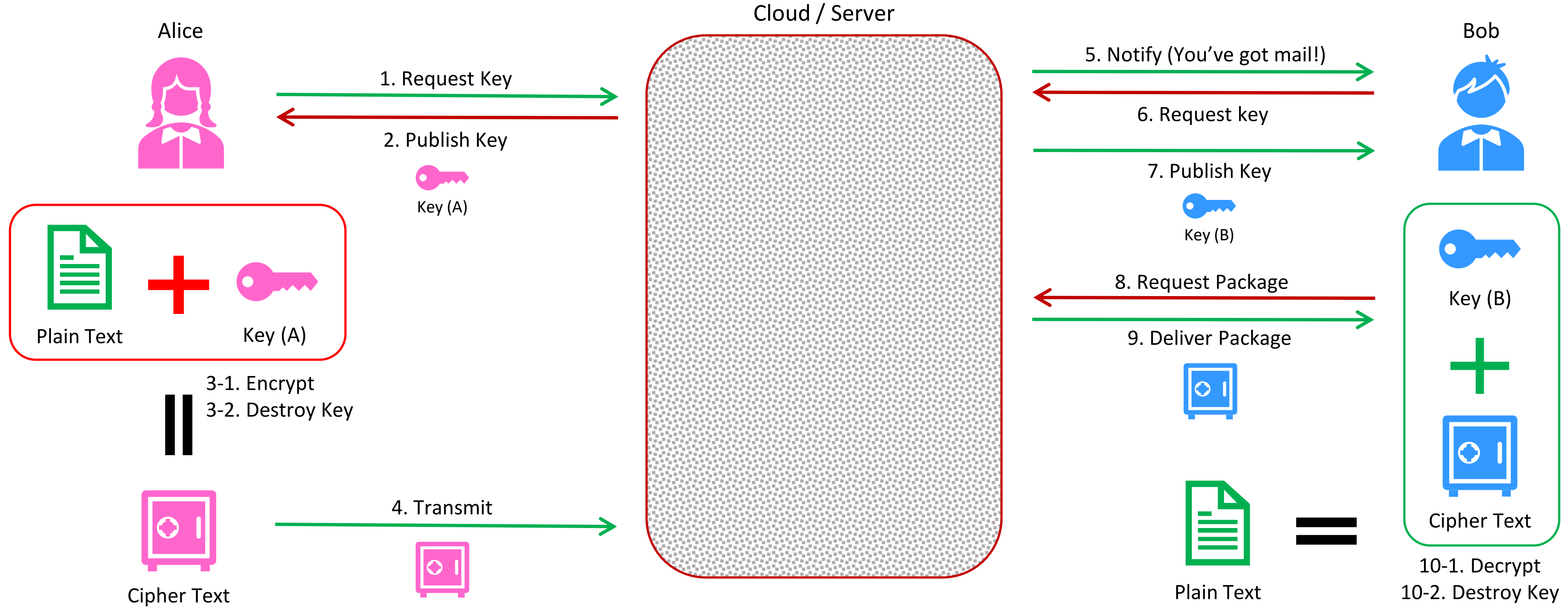
- Key-morphing is about 10% slower than normal encryption but is still faster than decryption.
  - Encryption = 171.43 MB/sec.
  - Decryption = 110.86 MB/sec.
  - Key-morphing = 153.27 MB/sec. (≈ 9 – 12% slower than normal encryption)
- Still faster than normal process of decrypting and re-encrypting.
  - Encryption time + decryption time = 8.51 seconds > 7.48 seconds = key-morphing time

How can morphing be … useful?



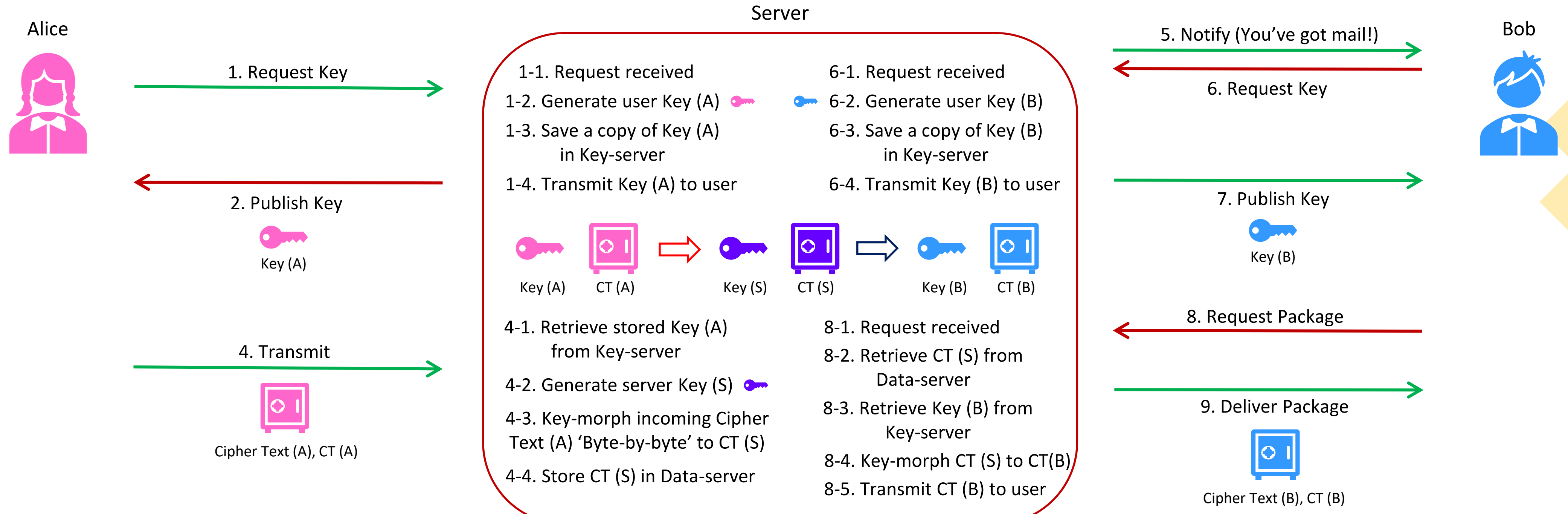
- The most powerful use will be the handling data at the lower communication channel.
- As the data pass the communication port, Vesper can morph the data into the new cipher text as they come through.
  - The original cipher text will NOT be stored anywhere in the receiving system.
  - Invalidates the original ‘user’ key
- The ‘user’ key becomes a temporary ‘one-time-use’ key.
  - The key in the receiver will be stored locally – <u>easy and safe</u> key management.

Vesper's Overall Communication Flow: User's View



- Works just like a normal asymmetric key system
  - Alice and Bob use different keys
- Both '1-time-use-only' keys are issued by the server(s)
  - Keys are always generated upon request and have time limit (expiration), and no two keys are the same.
  - Publishing keys require the separate account/user validation process (which is NOT shown above).

Overall Server-side Internal Process



1-1. Request received
1-2. Generate user Key (A)
1-3. Save a copy of Key (A) in Key-server
1-4. Transmit Key (A) to user

6-1. Request received
6-2. Generate user Key (B)
6-3. Save a copy of Key (B) in Key-server
6-4. Transmit Key (B) to user

4-1. Retrieve stored Key (A) from Key-server
4-2. Generate server Key (S)
4-3. Key-morph incoming Cipher Text (A) 'Byte-by-byte' to CT (S)
4-4. Store CT (S) in Data-server

8-1. Request received
8-2. Retrieve CT (S) from Data-server
8-3. Retrieve Key (B) from Key-server
8-4. Key-morph CT (S) to CT(B)
8-5. Transmit CT (B) to user

- Server's internal data, *'Key (S)' and the cipher text 'CT (S)'*, never leaves the server.
  - Upon *Key-morph*, cipher text NEVER gets decrypted.
  - Key-morph occurs byte-by-byte with no artifact. Even incoming and outgoing cipher texts (CT (A) and CT (B)) are NEVER saved in the server.
  - External keys (Key(A) and Key(B)) are 'one-time-use-only' and will not be accepted by server ever again.
- Maximum security is guaranteed.
  - Upon detecting any unusual activity, the entire data in the server will be *Key-morphed* instantly, invalidating 'all' keys existed before Key-morphing.
  - Any user (including internal personnel) logs off, all data accessed/used by that user account will be automatically *Key-morphed*.
  - Key-servers are physically separated into multiple servers, and each key will be randomly masked and stored separately to prevent attack, intrusion, and/or leak.

Illusion about size

The real-world cost of extra bytes wasted

16-bit vs. 32-bit Cipher Text

- Modulo operation is used to keep the value within a certain size, such as 16-bit or 32-bit.
  - Congruent modulo (a.k.a. remainder, mod, '%' operation) is always used to prevent values go over the limit.
  - It 'wraps' around the value: it defines the 'ring' set: Z/nZ = { 0, 1, …, n – 1 }
    - For example, Z/5Z = { 0, 1, 2, 3, 4 }: 13 (mod 5) = 3, 14 (mod 5) = 4, 15 (mod 5) = 0, …
- What is the meaning of comparing 16-bit cipher text and 32-bit cipher text?
  - Larger the size of the unit data, broader/wider range of data can be represented. Thus, if a simple 'guessing' the plain text based on cipher text, then, yes, 32-bit will lead to make more wrong guesses.
  - Why 128-bit RSA complexity is said to be (at most) 2^{128} when guessing one 32-bit value already costs 2^{32}? It is because that it is better to figure out the 'key' then guessing data (cipher text).
    - Simple guessing is a 'probability': To guess 1 out of 32-bit value may take 2^{32} guesses. It will take up to 2^{320} guesses to find out ten 32-bit values.
  - Cryptanalysis is either (1) to crack the key or (2) to find a correlation between known data to closely approximate the next one so that it will make 'logical' consequent guess and eventually will figure out all cipher texts.
- If we don't do 'brute-force' to solve the key, the next best thing for the cryptanalysis is finding out the pattern (correlation) among cipher text data.

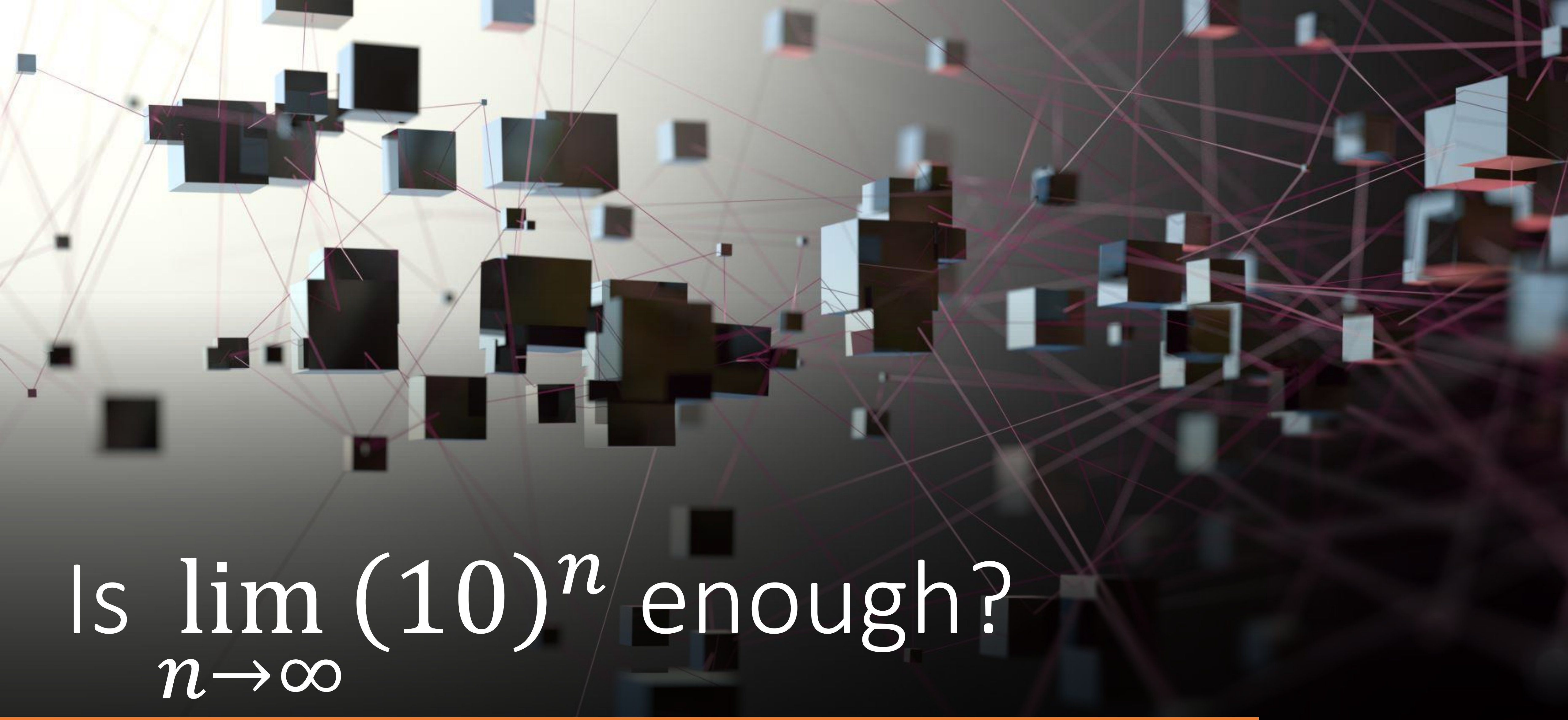
Making It Vague

- Which one is easier to find out the pattern?

0	D	A	C	0	0	E	B	0	0	0	F	C	0	0	0

0	0	A	0	0	0	0	B	0	0	0	0	C	0	0	0	0	D	0	0	0	0	E	0	0	0	0	F	0	0	0	0

- When data are present in a larger and more sparse space, the pattern stands out more obviously.
- If possible, it is better to use the <u>denser structure</u> for ambiguity and vagueness.
- Smaller data footprint means less memory used, and it also leads to the faster computation.
  - Calculation speeds between integers, floating points, and double precision floating points are dramatically different.
  - Memory IO (read/write from/to memory) for larger sized data is also take longer than smaller sized data.
  - Encryption/decryption requires billions and billions of calculations and IO: accumulates into a big performance difference.
- Prototype Vesper uses 16-bit data type for cipher text. It does not mean that 32-bit cannot be used; it is just that 16-bit is sufficient for the algorithm while it makes cryptanalysis/attacking harder.
- Therefore, simply comparing 16-bit and 32-bit cipher texts does not have much value. Each cipher can choose whatever more suitable type for its purpose.



Is $\lim_{n\to\infty}(10)^n$ enough?

How complex do you want it to be?

How about as complex as you want it to be?

Components for Vesper

- For each vector axis:
  - Vesper can use ANY NUMBER of '*axis coefficients*' for each axis. (Minimum of 2, no limit on maximum)
    - The multiple of those numbers determines the length of that axis.
      - For example, if 5 and 12 are used for coefficients of an axis, then the length of the axis is 60 (= 5 × 12).
  - One of the coefficients may NOT be a prime and can even be an EVEN NUMBER. → This is a ***game changer***!
    - Most, if not ALL, legacy RSA-based ciphers use two prime numbers.
      - Finding out one prime factor takes hundreds of years (and getting shorter); When one is found, the other is also found.
    - Vesper uses prime number(s) and a non-prime natural number. So, finding one prime does not guarantee anything due to the combinatorial problem.
      - E.g., let's say that 7 and 15 are used for two axis coefficients, and an attacker somehow finds out the size of the axis is 105. Since 105 can be factorized into 3 × 5 × 7, there are three possible combinations to figure out coefficients: (3 and 5), (3 and 7), and (5 and 7).
      - As coefficients get bigger and/or more than two coefficients are used, the number of possible combinations uncontrollably explodes. (Look for 'Little Omega Prime Function')
- Vesper uses the '*lattice*/*grid*' consist of at least two axes.
  - Based on the prototype Vesper setup, the size of the 'unit lattice' is:
    - Between 10^9 and 10^{15} using two axes with two coefficients (up to 1.263738E+16)
    - Between 10^{12} and 10^{23} using three axes with two coefficients (up to 1.4206458E+24)
  - Vesper is a configurable cipher by design. Coefficients and dimensional axes can be added for more complexity.
- These values are *just for the lattice setup* without actual vector! (On to the next page, please.)

Moving Vectors in Vesper

- Vesper uses vectors with random movements in the base grid/lattice.
  - Prototype employs at least 40-byte worth of random factors/coefficients to determine the movement of each vector.
    - Complexity is about 2.135987E+96.
  - There is NO possibility for any two vectors to use same movement pattern in Vesper algorithm.
    - Vesper uses well-known algorithm to make sure that no two vectors share the same pattern.
  - Vesper does not use any ECB (electronic code book, a.k.a. lookup table).
    - Along with the use of non-prime factor for the axis coefficient, no known attacking methods (including clear-text and chosen-cipher text) for RSA cipher will work.
    - The brute-force attack method may work only for a single key. Vesper can have at least 2^{8192} possible variations for keys.
  - Vesper is a ‘byte’ based cipher – not a block cipher. Thus, a formula for a single vector may not necessarily work for the next.
- Prototype (with the minimum setup) Vesper formula itself has average complexity of 2.6992E+112.
  - AES-256 is about 1.15792E+77.
  - This does NOT include the complexity added by applying pseudo-random numbers (like *Vigenère* cipher) as well as the combinations of axes coefficients from prime factorization of each axis.
  - Just from the implementation of the prototype Vesper, its complexity far exceeds that of RSA and AES-256.
  - Prototype Vesper uses two axes lattice with each axis employs two coefficients.

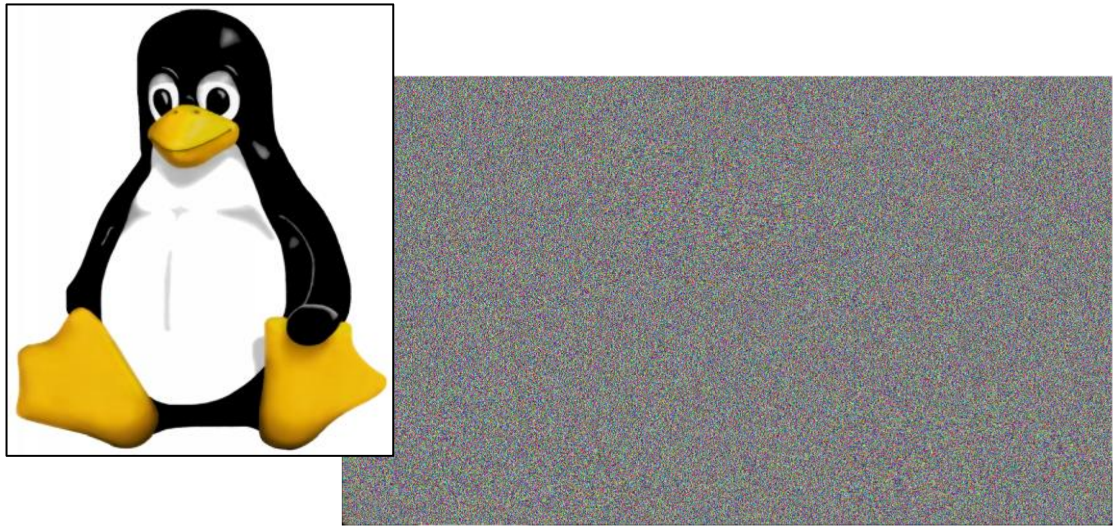
Truth about Security

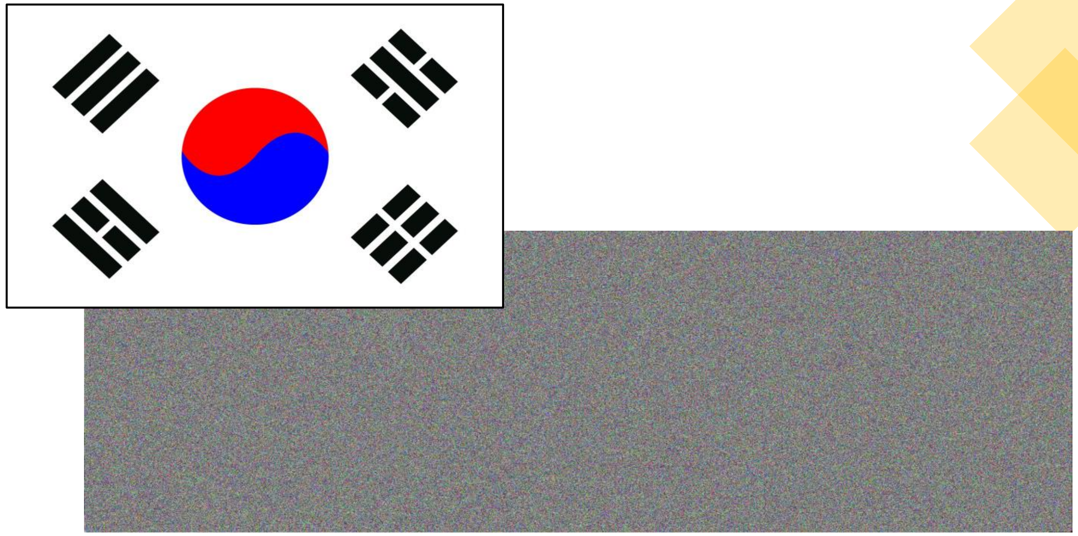
How secure is the Vesper against cryptanalysis/attack?

Security Features of Vesper

- Vesper uses 1024-byte key, which is 8192-bit key system.
  - It is larger than AES-256: about 2^{7936} times larger.
  - Brute-force key attack against Vesper is, as a matter of fact, hopeless.
    - It will take 7×10^{25} years for the Bitcoin network (yeah, THAT network) to crack the AES-128 key.
    - It will take 2.29×10^{32} years for the largest quantum computer (as of 2020 with 65 qubits) to crack the AES-256 key.
- Vesper is designed to be configurable.
  - Axis coefficients with the use of even number increases the level of security by introducing a combinatorial problem.
  - Making multi-dimensional (axes) lattice makes axis coefficient combination problem even harder to solve.
  - Number of axes coefficients and dimensional depth can be added and/or subtracted based on customer's needs/requirements.
- Vesper uses vectors moving on the lattice/grid.
  - The movement formula includes many coefficients/variable that are randomly calculated based on key.
  - Vector formula for the prototype Vesper already has the complexity of 10^{26}, and it does not include other complexities mentioned above.
- On top of all, Vesper uses internal scramble algorithm to further enhance the security.
  - The anti-temper for the key is already used; the anti-temper for the data will be added near future.
  - Pseudo-random XOR mechanism, like *Vigenère cipher*, is also used.
- Conclusion: The cryptanalysis/attack against Vesper is near impossible.

See them for yourself





- No trace of ECB (Electronic Code Book)
- Strong scrambling: combinatory (random + Vigenère) masking

(If you still can see a penguin or the flag, you really need to see your doctor, seriously.)

Cryptanalysis: Dimensional Coefficients

- Vesper Lattice: the most enigmatic feature
  - Vesper lattice can have any dimension more than 2. (Vesper can have, say, 20 dimension. Well, theoretically.)
- Each dimension can have any number of dimension coefficients.
  - E. g., for 2 dimension, each axis (x and y) has different dimensional coefficients.
  - The multiplication of all dimensional coefficients for a single axis determines the size of that axis.
  - Dimensional coefficients must be relative primes, and one can be an even number.
    - Totally different from RSA's two co-primes that is governed and proved to be volatile from Shor's algorithm for quantum computing.
- Finding out the dimensional coefficients for all axes is crucial key to attack Vesper. However,
  - The solution is harder than 'subset sum' problem, which is NP-hard.
  - The first step is to determine the length of the axis. Then, finding out all dimensional coefficients is, well, amazingly time-consuming.
    - E.g.: If the length is 770, the divisors are 2, 5, 7, and 11. If attacker finds out there are three dimensional coefficients, the possible combinations are:

      (2, 385), (5, 154), (7, 110), (11, 70), (10, 77), (14, 55), (22, 35) → Try for actual numbers like 1,001,417,796,266.

Cryptanalysis: Random Number Masking

- Vesper utilizes two step masking process.
  - 1st Step: Random number masking
    - For demo, there are more than 10 key numbers + same number of random number masks
    - This is to scramble and to obfuscate the original data (plain text).
  - 2nd Step: Vigenère masking
    - Vigenère encryption algorithm is proven to be 'impossible' to solve mathematically.
    - This step is to prevent and incapacitate frequency analysis against Vesper cipher text.
- There is no known 'logical' method to find out the random numbers.
- Vesper is NOT an RSA-based algorithm and is NOT covered by Shor's algorithm.
  - The only way to attack Vesper is the brute-force attack.

==

- UPDATE: Vesper encryption key is now (from version 1.3) 1024-byte (1MB) and provide:
  - Various license types and expiration date
  - Key-morphing validity checking
  - Origin check
  - Transfer of ownership
  - Secure key-validation (SHA-256)
  - and more!

Vesper

vs.

Homomorphic Encryption

Are they like apples and oranges?

Why does HE appear bigger than what it is?

Ideal vs. Reality

- Difference between ‘academia’ and ‘real-world’
  - ‘Infinity’ does not exist in the real-world: ALL numbers must be represented in the given data size (16-bit, 32-bit, etc.).
    - Larger subset of the real numbers (floating-point numbers) cannot be represented in computers, by design.
    - I.e. 0.1 is stored as 0.100000001490116119384765625 in 32-bit (IEEE-754).
  - In other words, the fact that *’representing numbers in the computer’* itself *introduces some errors* already.
  - Homomorphic Encryption (HE) intentionally injects ‘errors’, called ‘noise’, to enhance the security against cryptanalysis.
    - How can we decide that how much ‘noise’ is enough? Are there certain criteria for these ‘noise’ to satisfy requirements for ALL cases?
  - Under HE, the noise accumulates over time, and, thus, HE performs a procedure called ‘bootstrap’ to keep noise down.
  - Extra process requires additional resources (i.e., memory and time) in real-world.
    - Manipulation of data (e.g., multiplying some factors, bootstrapping, etc.) to reduce the errors may look simple in theory on a paper, but each process requires additional resources for computers to store, to read, to write, and to calculate data. Moreover, as data gets larger, more resources are needed.
  - Assurance against cryptanalysis: What happens when an attacker introduces larger errors to the data?
    - How can HE find the abnormal level of errors in the encrypted data? Can bootstrapping resolve the error and keep the error under control?
    - If HE detects any irregularity of the data, how does HE know if the error is intentional by customers or abnormal by attacker?
- Homomorphic encryption is a great field for further research and study.
  - However, it does not necessarily mean that it is the most suitable for the applications, at least for the world as of today.
    - The real-world applications must consider the performance, resources, and maintainability/sustainability.

In Reality: Why is Vesper for the real-world?

- Performance: Vesper is ***super fast***!
  - Testing shows that the prototype (without any performance optimization) encrypts at 160+ MB/sec and decrypts at 100+ MB/sec.
    - Tests were performed on the machine with Intel® i9-12900KF @ 3.19GHz, normal priority under Windows 11.
    - The speed is fast enough to encrypt/decrypt 4K HD movies in real-time even without any optimization.
- No injected error or additional manipulations
  - The calculation results are based on the exact data as they are provided without any hidden errors or adjustments.
  - It is user to decide what to accept and how to interpret the result – not for computers or algorithms to do human's work.
- Ability to calculate and evaluate the encrypted data in cipher text form
  - Not only addition and multiplication, but Vesper can also perform validation (>, >=, !=, <=, <) without decryption of data.
- Ability to search/match for the data in cipher text form
  - Text search and data matching can be done without decryption of data.
- Small size for the cipher text
  - Basic configuration only 'doubles' (×2) the size after encryption.
    - Using minimum numbers for the axes and dimension configuration.
    - The size will increase as the number of dimensions and dimension factors increase.
  - Additional dimensions and axes coefficients will increase the size; however, it is fixed amount and is still much smaller than HE.

Anonymity? Needs More Than Cipher Algorithm

- Keeping the personal information safe is <u>NOT</u> a job for the cipher algorithms entirely.
    - Data need to be carefully structured and organized, and the database also needs to be carefully designed.
        - Encrypt personal data into a file with the strongest known cipher then naming it with its owner's name does not keep the anonymity under cover.
    - The most important thing for the anonymity is how to structure/manage the data – not which cipher algorithm to use.
        - 'Encrypting name and not encrypting his/her DNA data' is the same as 'not encrypting name and encrypting DNA data.'
- The servers and their architecture must be designed and setup to keep the information safe.
    - Without a gun, the bullet is just a piece of metal, no matter how great the bullet is.
    - Cipher algorithm must provide certain features to keep servers secure and efficient while fast for service.
        - Vesper is the only solution for the information security: for data itself AND for the ironclad server design.
        - Key-morphing feature of Vesper satisfies:
            - Total isolation of internal data
            - Limiting the keys to be valid only for the external data
            - Instantly invalidating leaked keys
            - Hiding the matching keys from data
        - Cipher text searching capability of Vesper enables servers to keep data and key indexes always encrypted.
            - Intruders cannot do anything as the information is encrypted and useless without correct key.
            - Even if hacker steals the data, server can perform key-morphing to change the key and data instantly.
    - All those capabilities need to be performed by servers but not by the cipher algorithm. It is a management matter.

Performance

Numbers That Matter

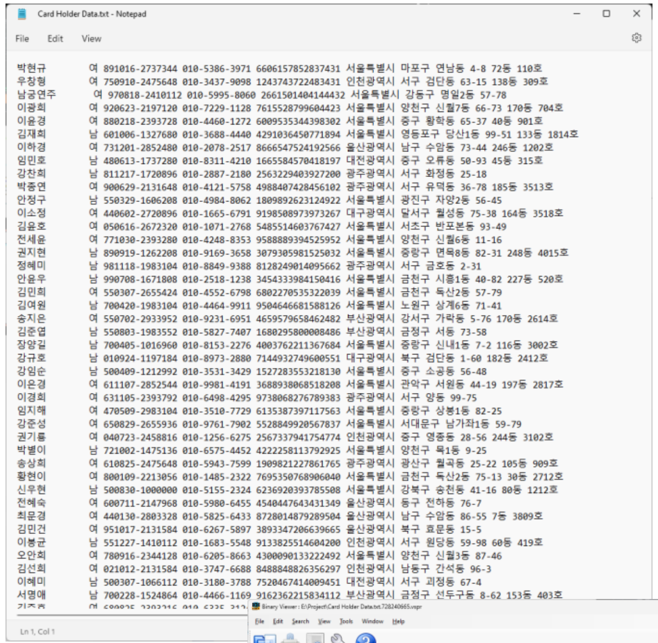
Average Performance Summary

• Encryption:	160 – 170 MB/sec.
• Decryption:	100 – 110 MB/sec.
• Key-morphing:	140 – 150 MB/sec.
• Simple search:	2,000,000 fields/sec.
• RDBMS query:	3,000,000 fields/sec. for 7 queries

- Result recording and calculation are done by Vesper core library.
- RDMBS configuration file parsing and Vesper component configuration are performed by the demo program.

- NOTE:
  - Test result may vary based on the system configuration.
  - Tests were performed on the machine with Intel® i9-12900KF @ 3.19GHz, normal priority under Windows 11 Pro 64-bit.

Vesper RDBMS Demo & Performance



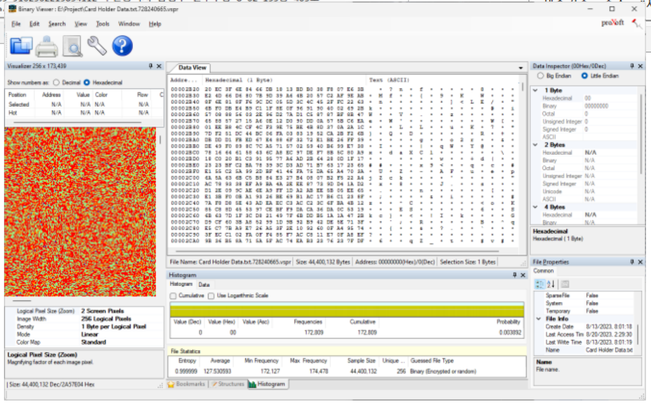
Card Holder Data.txt - Notepad

```
박현규        여 891016-2737344 010-5386-3971 6606157852837431 서울특별시 마포구 연남동 4-8 72동 110호
우창형        여 750910-2475648 010-3437-9098 1243743722483431 인천광역시 서구 검단동 63-15 138동 309호
남궁연주       여 970818-2410112 010-5995-8060 2661501404144432 서울특별시 강동구 명일2동 57-78
이광희        여 920623-2197120 010-7229-1128 7615528799604423 서울특별시 양천구 신월7동 66-73 170동 704호
이윤경        여 880218-2393728 010-4460-1272 6009535344398302 서울특별시 중구 황학동 65-37 40동 901호
김재희        남 601006-1327680 010-3688-4440 4291036450771894 서울특별시 영등포구 당산1동 99-51 133동 1814호
이하경        여 731201-2852480 010-2078-2517 8666547524192566 울산광역시 남구 수암동 73-44 246동 1202호
임민호        남 480613-1737280 010-8311-4210 1665584570418197 대전광역시 중구 오류동 50-93 45동 315호
강찬희        남 811217-1720896 010-2887-2180 2563229403927200 광주광역시 서구 화정동 25-18
박종연        여 900629-2131648 010-4121-5758 4988407428456102 광주광역시 서구 유덕동 36-78 185동 3513호
안정구        남 550329-1606208 010-4984-8062 1809892623124922 서울특별시 광진구 자양2동 56-45
이소정        여 440602-2720896 010-1665-6791 9198508973973267 대구광역시 달서구 월성동 75-38 164동 3518호
김윤호        여 050616-2672320 010-1071-2768 5485514603767427 서울특별시 서초구 반포본동 93-49
전세윤        여 771030-2393280 010-4248-8353 9588889394525952 서울특별시 양천구 신월6동 11-16
권지현        남 890919-1262208 010-9169-3658 3079305981525032 서울특별시 중랑구 면목8동 82-31 248동 4015호
정혜미        남 981118-1983104 010-8849-9388 8128249014095662 광주광역시 서구 금호동 2-31
안윤우        남 990708-1671808 010-2518-1238 3454333984150416 서울특별시 금천구 시흥1동 40-82 227동 520호
김민희        여 550307-2655424 010-4552-6798 6802270535322039 서울특별시 금천구 독산2동 57-79
김여원        남 700420-1983104 010-4464-9911 9504646681588126 서울특별시 노원구 상계6동 71-41
송지은        여 550702-2933952 010-9231-6951 4659579658462482 부산광역시 강서구 가락동 5-76 170동 2614호
김준엽        남 550803-1983552 010-5827-7407 1680295800008486 부산광역시 금정구 서동 73-58
장양길        남 700405-1016960 010-8153-2276 4003762211367684 서울특별시 중랑구 신내1동 7-2 116동 3002호
강규호        남 010924-1197184 010-8973-2880 7144932749600551 대구광역시 북구 검단동 1-60 182동 2412호
강임순        남 500409-1212992 010-3531-3429 1527283553218130 서울특별시 중구 소공동 56-48
이은경        여 611107-2852544 010-9981-4191 3688938068518208 서울특별시 관악구 서원동 44-19 197동 2817호
이경희        여 631105-2393792 010-6498-4295 9738068276789383 광주광역시 서구 양동 99-75
임지해        여 470509-2983104 010-3510-7729 6135387397117563 서울특별시 중랑구 상봉1동 82-25
강준성        여 650829-2655936 010-9761-7902 5528849920567837 서울특별시 서대문구 남가좌1동 59-79
권기룡        여 040723-2458816 010-1256-6275 2567337941754774 인천광역시 중구 영종동 28-56 244동 3102호
박별이        남 721002-1475136 010-6575-4452 4222258113792925 서울특별시 양천구 목1동 9-25
송상희        여 610825-2475648 010-5943-7599 1909821227861765 광주광역시 광산구 월곡동 25-22 105동 909호
황현이        여 800109-2213056 010-1485-2322 7695350768906040 서울특별시 금천구 독산2동 75-13 30동 2712호
신우현        남 500830-1000000 010-5155-2324 6236920393785508 서울특별시 강북구 송천동 41-16 80동 1212호
전혜숙        여 600711-2147968 010-5980-6455 4540447643431349 울산광역시 동구 전하동 76-7
최문경        여 440130-2803328 010-5825-6433 8728014879289504 울산광역시 남구 수암동 86-55 7동 3809호
김민건        여 951017-2131584 010-6267-5897 3893347206639665 울산광역시 북구 효문동 15-5
이봉균        남 551227-1410112 010-1683-5548 9133825514604200 인천광역시 서구 원당동 59-98 60동 419호
오안희        여 780916-2344128 010-6205-8663 4300090133222492 서울특별시 양천구 신월3동 87-46
김선희        여 021012-2131584 010-3747-6688 8488848826356297 인천광역시 남동구 간석동 96-3
이혜미        남 500307-1066112 010-3180-3788 7520467414009451 대전광역시 서구 괴정동 67-4
서명애        남 700228-1524864 010-4466-1169 9162362215834112 부산광역시 금정구 선두구동 8-62 153동 403호
```

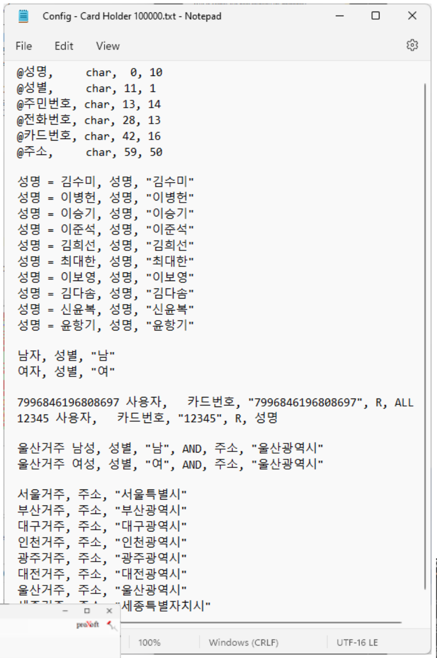
Ln 1, Col 1

100,000 People Data (generated)

Encrypted



Binary Viewer : E:\Project\Card Holder Data.txt.728240665.vspr
File Edit Search View Tools Window Help
Visualizer 256 x 173,439
Data View
Data Inspector (00Hex/0Dec)
Histogram
File Properties



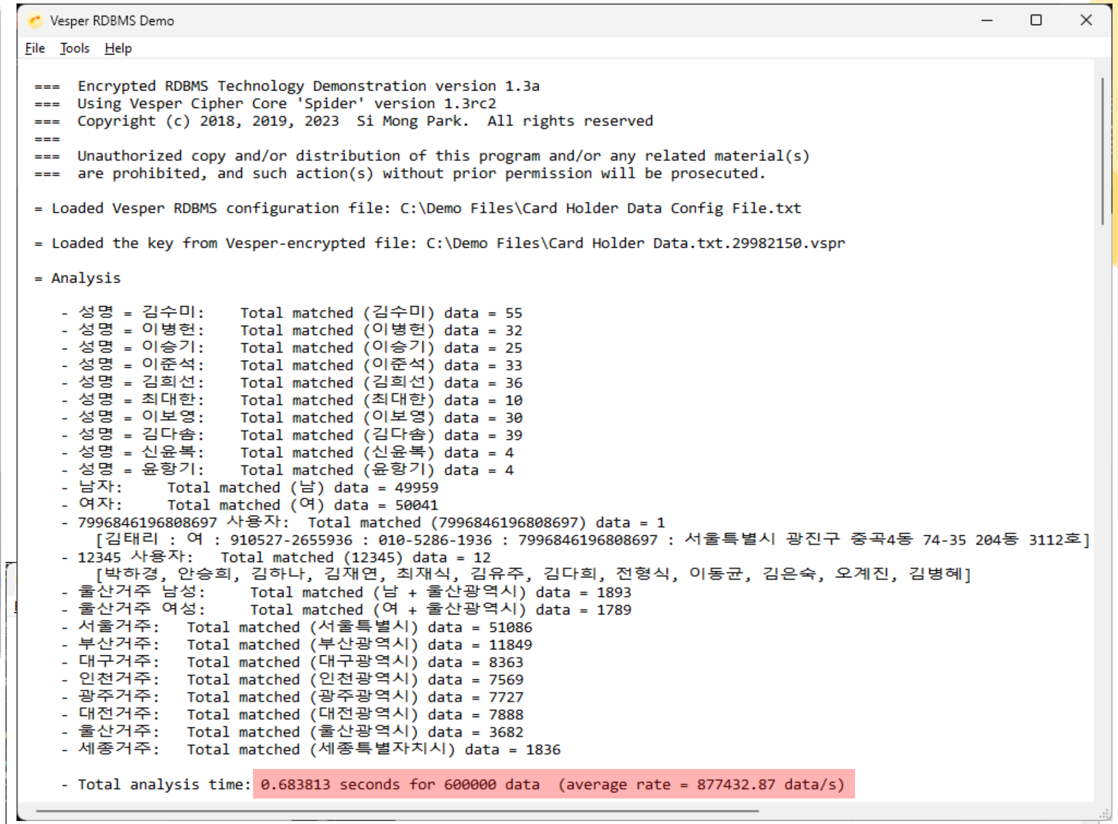
Config - Card Holder 100000.txt - Notepad

```
@성명,     char,  0, 10
@성별,     char, 11, 1
@주민번호, char, 13, 14
@전화번호, char, 28, 13
@카드번호, char, 42, 16
@주소,     char, 59, 50

성명 = 김수미, 성명, "김수미"
성명 = 이병헌, 성명, "이병헌"
성명 = 이승기, 성명, "이승기"
성명 = 이준석, 성명, "이준석"
성명 = 김희선, 성명, "김희선"
성명 = 최대한, 성명, "최대한"
성명 = 이보영, 성명, "이보영"
성명 = 김다솜, 성명, "김다솜"
성명 = 신윤복, 성명, "신윤복"
성명 = 윤향기, 성명, "윤향기"

남자, 성별, "남"
여자, 성별, "여"

7996846196808697 사용자,   카드번호, "7996846196808697", R, ALL
12345 사용자,   카드번호, "12345", R, 성명

울산거주 남성, 성별, "남", AND, 주소, "울산광역시"
울산거주 여성, 성별, "여", AND, 주소, "울산광역시"

서울거주, 주소, "서울특별시"
부산거주, 주소, "부산광역시"
대구거주, 주소, "대구광역시"
인천거주, 주소, "인천광역시"
광주거주, 주소, "광주광역시"
대전거주, 주소, "대전광역시"
울산거주, 주소, "울산광역시"
"세종특별자치시"
```

100%   Windows (CRLF)   UTF-16 LE

Simple Configuration

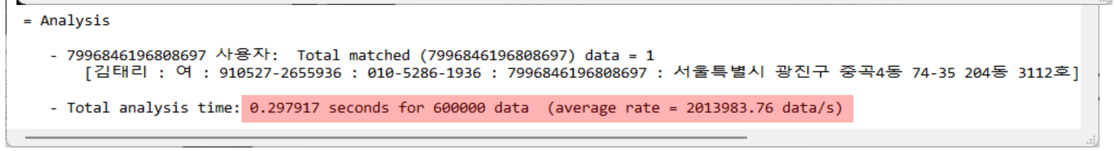
Analysis results for 24 rules for 100,000 people (6 fields per person)



Vesper RDBMS Demo

File Tools Help

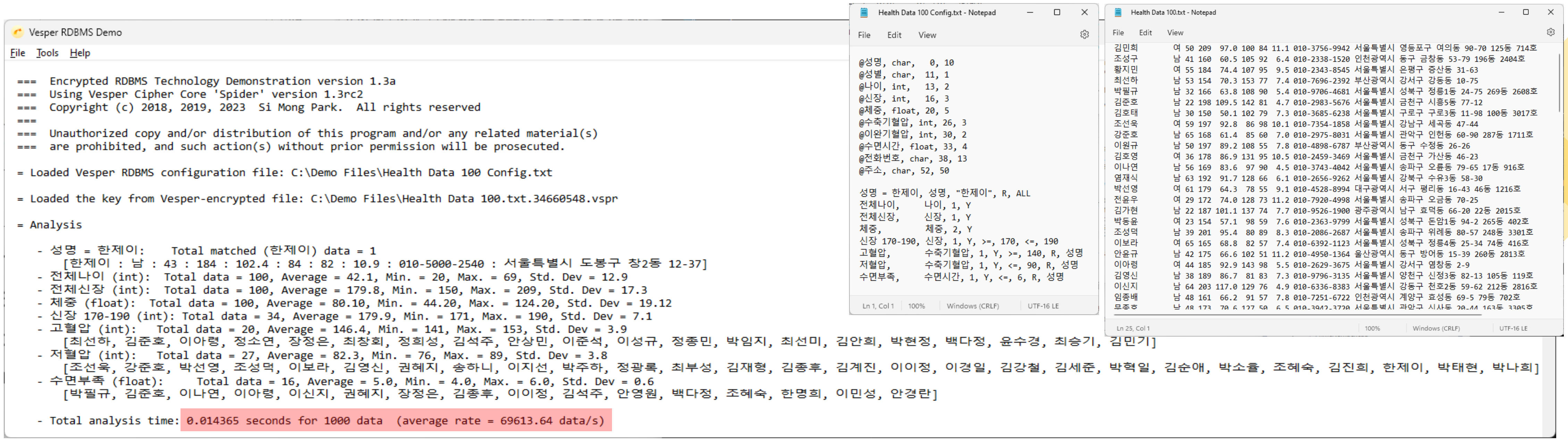
```
===  Encrypted RDBMS Technology Demonstration version 1.3a
===  Using Vesper Cipher Core 'Spider' version 1.3rc2
===  Copyright (c) 2018, 2019, 2023  Si Mong Park.  All rights reserved
===
===  Unauthorized copy and/or distribution of this program and/or any related material(s)
===  are prohibited, and such action(s) without prior permission will be prosecuted.

= Loaded Vesper RDBMS configuration file: C:\Demo Files\Card Holder Data Config File.txt

= Loaded the key from Vesper-encrypted file: C:\Demo Files\Card Holder Data.txt.29982150.vspr

= Analysis

  - 성명 = 김수미:   Total matched (김수미) data = 55
  - 성명 = 이병헌:   Total matched (이병헌) data = 32
  - 성명 = 이승기:   Total matched (이승기) data = 25
  - 성명 = 이준석:   Total matched (이준석) data = 33
  - 성명 = 김희선:   Total matched (김희선) data = 36
  - 성명 = 최대한:   Total matched (최대한) data = 10
  - 성명 = 이보영:   Total matched (이보영) data = 30
  - 성명 = 김다솜:   Total matched (김다솜) data = 39
  - 성명 = 신윤복:   Total matched (신윤복) data = 4
  - 성명 = 윤향기:   Total matched (윤향기) data = 4
  - 남자:   Total matched (남) data = 49959
  - 여자:   Total matched (여) data = 50041
  - 7996846196808697 사용자:  Total matched (7996846196808697) data = 1
      [김태리 : 여 : 910527-2655936 : 010-5286-1936 : 7996846196808697 : 서울특별시 광진구 중곡4동 74-35 204동 3112호]
  - 12345 사용자:  Total matched (12345) data = 12
      [박하경, 안승희, 김하나, 김재연, 최재식, 김유주, 김다희, 전형식, 이동균, 김은숙, 오계진, 김병혜]
  - 울산거주 남성:   Total matched (남 + 울산광역시) data = 1893
  - 울산거주 여성:   Total matched (여 + 울산광역시) data = 1789
  - 서울거주:  Total matched (서울특별시) data = 51086
  - 부산거주:  Total matched (부산광역시) data = 11849
  - 대구거주:  Total matched (대구광역시) data = 8363
  - 인천거주:  Total matched (인천광역시) data = 7569
  - 광주거주:  Total matched (광주광역시) data = 7727
  - 대전거주:  Total matched (대전광역시) data = 7888
  - 울산거주:  Total matched (울산광역시) data = 3682
  - 세종거주:  Total matched (세종특별자치시) data = 1836

  - Total analysis time: 0.683813 seconds for 600000 data  (average rate = 877432.87 data/s)
```



```
= Analysis

  - 7996846196808697 사용자:  Total matched (7996846196808697) data = 1
      [김태리 : 여 : 910527-2655936 : 010-5286-1936 : 7996846196808697 : 서울특별시 광진구 중곡4동 74-35 204동 3112호]

  - Total analysis time: 0.297917 seconds for 600000 data  (average rate = 2013983.76 data/s)
```

Analysis result for 1 rule for 100,000 people (6 fields per person)

Performance Shootout



Vesper RDBMS Demo

```
===  Encrypted RDBMS Technology Demonstration version 1.3a
===  Using Vesper Cipher Core 'Spider' version 1.3rc2
===  Copyright (c) 2018, 2019, 2023  Si Mong Park.  All rights reserved
===
===  Unauthorized copy and/or distribution of this program and/or any related material(s)
===  are prohibited, and such action(s) without prior permission will be prosecuted.

= Loaded Vesper RDBMS configuration file: C:\Demo Files\Health Data 100 Config.txt

= Loaded the key from Vesper-encrypted file: C:\Demo Files\Health Data 100.txt.34660548.vspr

= Analysis

  - 성명 = 한제이:    Total matched (한제이) data = 1
     [한제이 : 남 : 43 : 184 : 102.4 : 84 : 82 : 10.9 : 010-5000-2540 : 서울특별시 도봉구 창2동 12-37]
  - 전체나이 (int):  Total data = 100, Average = 42.1, Min. = 20, Max. = 69, Std. Dev = 12.9
  - 전체신장 (int):  Total data = 100, Average = 179.8, Min. = 150, Max. = 209, Std. Dev = 17.3
  - 체중 (float):  Total data = 100, Average = 80.10, Min. = 44.20, Max. = 124.20, Std. Dev = 19.12
  - 신장 170-190 (int): Total data = 34, Average = 179.9, Min. = 171, Max. = 190, Std. Dev = 7.1
  - 고혈압 (int):   Total data = 20, Average = 146.4, Min. = 141, Max. = 153, Std. Dev = 3.9
     [최선하, 김준호, 이아령, 정소연, 장정은, 최창회, 정희성, 김석주, 안상민, 이준석, 이성규, 정종민, 박임지, 최선미, 김안희, 박현정, 백다정, 윤수경, 최승기, 김민기]
  - 저혈압 (int):   Total data = 27, Average = 82.3, Min. = 76, Max. = 89, Std. Dev = 3.8
     [조선욱, 강준호, 박선영, 조성덕, 이보라, 김영신, 권혜지, 송하니, 이지선, 박주하, 정광록, 최부성, 김재형, 김종후, 김계진, 이이정, 이경일, 김강철, 김세준, 박혁일, 김순애, 박소율, 조혜숙, 김진희, 한제이, 박태현, 박나희]
  - 수면부족 (float):    Total data = 16, Average = 5.0, Min. = 4.0, Max. = 6.0, Std. Dev = 0.6
     [박필규, 김준호, 이나연, 이아령, 이신지, 권혜지, 장정은, 김종후, 이이정, 김석주, 안영원, 백다정, 조혜숙, 한명희, 이민성, 안경란]

  - Total analysis time: 0.014365 seconds for 1000 data  (average rate = 69613.64 data/s)
```

Configuration File

Health Data 100 Config.txt - Notepad

```
@성명, char,   0, 10
@성별, char,  11, 1
@나이, int,   13, 2
@신장, int,   16, 3
@체중, float, 20, 5
@수축기혈압, int, 26, 3
@이완기혈압, int, 30, 2
@수면시간, float, 33, 4
@전화번호, char, 38, 13
@주소, char, 52, 50

성명 = 한제이, 성명, "한제이", R, ALL
전체나이,     나이, 1, Y
전체신장,     신장, 1, Y
체중,         체중, 2, Y
신장 170-190, 신장, 1, Y, >=, 170, <=, 190
고혈압,       수축기혈압, 1, Y, >=, 140, R, 성명
저혈압,       수축기혈압, 1, Y, <=, 90, R, 성명
수면부족,     수면시간, 1, Y, <=, 6, R, 성명
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-16 LE

100 People Health Data (generated)

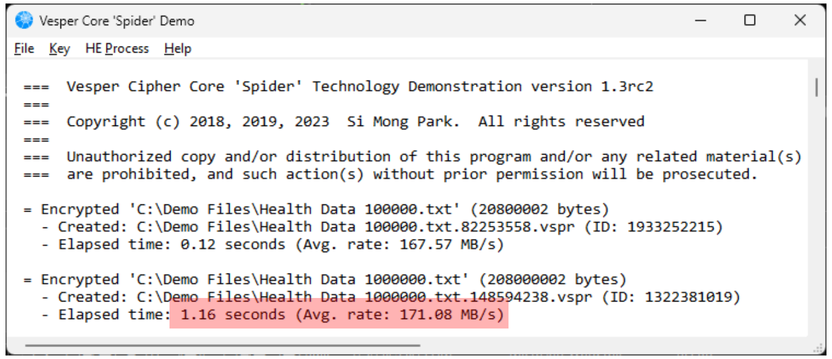
Health Data 100.txt - Notepad

```
김민희       여 50 209  97.0 100 84 11.1 010-3756-9942 서울특별시 영등포구 여의동 90-70 125동 714호
조성구       남 41 160  60.5 105 92  6.4 010-2338-1520 인천광역시 동구 금창동 53-79 196동 2404호
황지민       여 55 184  74.4 107 95  9.5 010-2343-8545 서울특별시 은평구 증산동 31-63
최선하       남 53 154  70.3 153 77  7.4 010-7696-2392 부산광역시 강서구 강동동 10-75
박필규       남 32 166  63.8 108 90  5.4 010-9706-4681 서울특별시 성북구 정릉1동 24-75 269동 2608호
김준호       남 22 198 109.5 142 81  4.7 010-2983-5676 서울특별시 금천구 시흥5동 77-12
김호태       남 30 150  50.1 102 79  7.3 010-3685-6238 서울특별시 구로구 구로3동 11-98 100동 3017호
조선욱       여 59 197  92.8  86 98 10.1 010-7354-1858 서울특별시 강남구 세곡동 47-44
강준호       남 65 168  61.4  85 60  7.0 010-2975-8031 서울특별시 관악구 인헌동 60-90 287동 1711호
이원규       남 50 197  89.2 108 55  7.8 010-4898-6787 부산광역시 동구 수정동 26-26
김호영       여 36 178  86.9 131 95 10.5 010-2459-3469 서울특별시 금천구 가산동 46-23
이나연       남 56 169  83.6  97 90  4.5 010-3743-4042 서울특별시 송파구 오륜동 79-65 17동 916호
염재식       남 63 192  91.7 128 66  6.1 010-2656-9262 서울특별시 강북구 수유3동 58-30
박선영       여 61 179  64.3  78 55  9.1 010-4528-8994 대구광역시 서구 평리동 16-43 46동 1216호
전윤우       여 29 172  74.0 128 73 11.2 010-7920-4998 서울특별시 송파구 오금동 70-25
김가현       남 22 187 101.1 137 74  7.7 010-9526-1900 광주광역시 남구 효덕동 66-20 22동 2015호
박동윤       여 23 154  57.1  98 59  7.6 010-2363-9799 서울특별시 성북구 돈암1동 94-2 265동 402호
조성덕       남 39 201  95.4  80 89  8.3 010-2086-2687 서울특별시 송파구 위례동 80-57 248동 3301호
이보라       여 65 165  68.8  82 57  7.4 010-6392-1123 서울특별시 성북구 정릉4동 25-34 74동 416호
안윤규       남 42 175  66.6 102 51 11.2 010-4950-1364 울산광역시 동구 방어동 15-39 260동 2813호
이아령       여 44 185  92.9 143 98  5.5 010-2629-3675 서울특별시 강서구 염창동 2-9
김영신       남 38 189  86.7  81 83  7.3 010-9796-3135 서울특별시 양천구 신정3동 82-13 105동 119호
이신지       남 64 203 117.0 129 76  4.9 010-6336-8383 서울특별시 강동구 천호2동 59-62 212동 2816호
임종배       남 48 161  66.2  91 57  7.8 010-7251-6722 인천광역시 계양구 효성동 69-5 79동 702호
```

Ln 25, Col 1 | 100% | Windows (CRLF) | UTF-16 LE

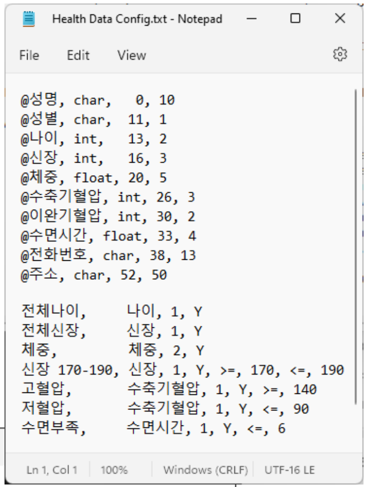
- Test used data of 100 people's health data, and each person's record contains 10 fields: name, sex, age, height, weight, systolic/diastolic blood pressure, sleep hours, phone number, and address.
  - All data are encrypted with Vesper 1.3rc2 – no data is stored decrypted.
  - No plaintext data preprocessing is applied; characters and numbers are mixed as shown above screen shot.
  - 8 rules total: 7 rules to calculate average and standard deviation along with maximum and minimum value records.
- Result: 0.014365 seconds (= 69,613.64 data per second)

Push the Limit with 1,000,000 People Health Data



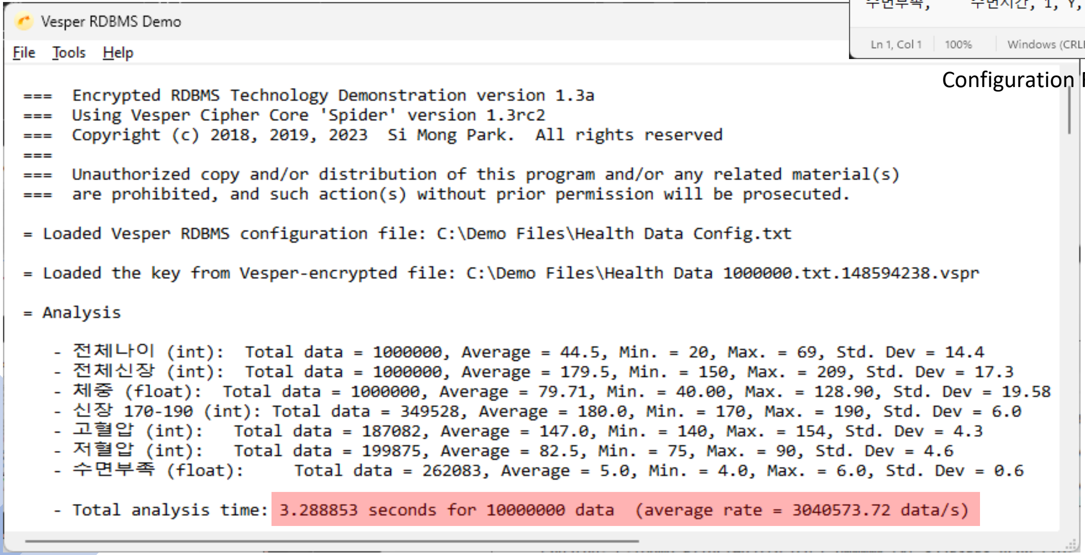
```
===  Vesper Cipher Core 'Spider' Technology Demonstration version 1.3rc2
===
===  Copyright (c) 2018, 2019, 2023  Si Mong Park.  All rights reserved
===
===  Unauthorized copy and/or distribution of this program and/or any related material(s)
===  are prohibited, and such action(s) without prior permission will be prosecuted.

= Encrypted 'C:\Demo Files\Health Data 100000.txt' (20800002 bytes)
  - Created: C:\Demo Files\Health Data 100000.txt.82253558.vspr (ID: 1933252215)
  - Elapsed time: 0.12 seconds (Avg. rate: 167.57 MB/s)

= Encrypted 'C:\Demo Files\Health Data 1000000.txt' (208000002 bytes)
  - Created: C:\Demo Files\Health Data 1000000.txt.148594238.vspr (ID: 1322381019)
  - Elapsed time: 1.16 seconds (Avg. rate: 171.08 MB/s)
```

Time for Encryption



```
@성명, char,   0, 10
@성별, char,  11, 1
@나이, int,   13, 2
@신장, int,   16, 3
@체중, float, 20, 5
@수축기혈압, int, 26, 3
@이완기혈압, int, 30, 2
@수면시간, float, 33, 4
@전화번호, char, 38, 13
@주소, char, 52, 50

전체나이,     나이, 1, Y
전체신장,     신장, 1, Y
체중,         체중, 2, Y
신장 170-190, 신장, 1, Y, >=, 170, <=, 190
고혈압,       수축기혈압, 1, Y, >=, 140
저혈압,       수축기혈압, 1, Y, <=, 90
수면부족,     수면시간, 1, Y, <=, 6
```

Configuration File

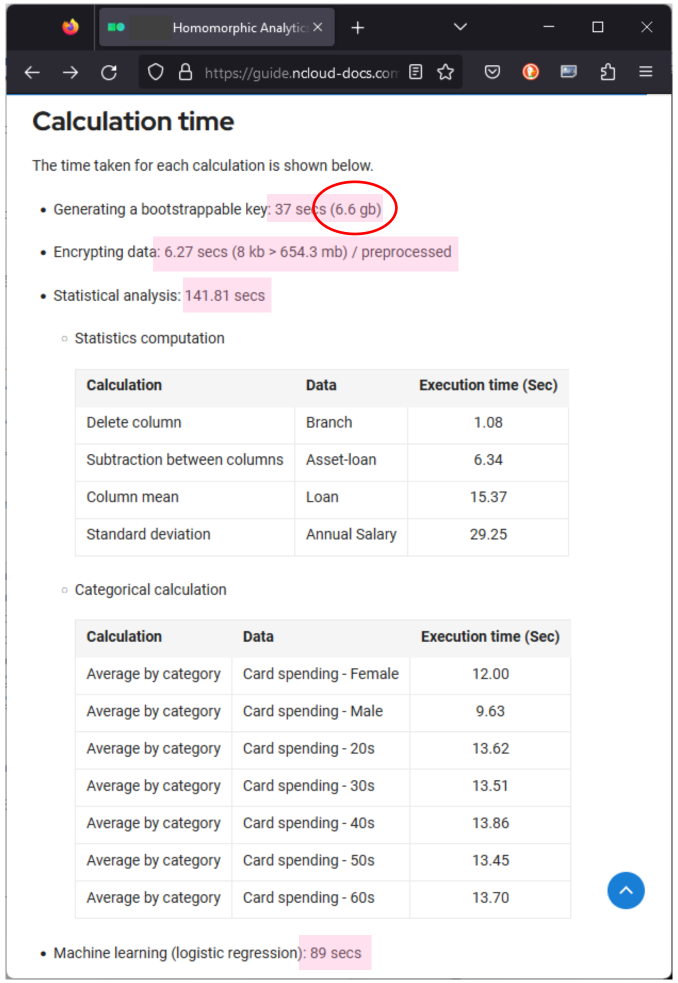


```
===  Encrypted RDBMS Technology Demonstration version 1.3a
===  Using Vesper Cipher Core 'Spider' version 1.3rc2
===  Copyright (c) 2018, 2019, 2023  Si Mong Park.  All rights reserved
===
===  Unauthorized copy and/or distribution of this program and/or any related material(s)
===  are prohibited, and such action(s) without prior permission will be prosecuted.

= Loaded Vesper RDBMS configuration file: C:\Demo Files\Health Data Config.txt

= Loaded the key from Vesper-encrypted file: C:\Demo Files\Health Data 1000000.txt.148594238.vspr

= Analysis

  - 전체나이 (int):  Total data = 1000000, Average = 44.5, Min. = 20, Max. = 69, Std. Dev = 14.4
  - 전체신장 (int):  Total data = 1000000, Average = 179.5, Min. = 150, Max. = 209, Std. Dev = 17.3
  - 체중 (float):  Total data = 1000000, Average = 79.71, Min. = 40.00, Max. = 128.90, Std. Dev = 19.58
  - 신장 170-190 (int): Total data = 349528, Average = 180.0, Min. = 170, Max. = 190, Std. Dev = 6.0
  - 고혈압 (int):   Total data = 187082, Average = 147.0, Min. = 140, Max. = 154, Std. Dev = 4.3
  - 저혈압 (int):   Total data = 199875, Average = 82.5, Min. = 75, Max. = 90, Std. Dev = 4.6
  - 수면부족 (float):     Total data = 262083, Average = 5.0, Min. = 4.0, Max. = 6.0, Std. Dev = 0.6

  - Total analysis time: 3.288853 seconds for 10000000 data  (average rate = 3040573.72 data/s)
```

Time for Analysis

- Performed analysis with 1,000,000 people's health data (generated).
  - 10 fields for each person
  - Total of 10,000,000 data fields
  - Data file size (encrypted) = 396MB (with key)
- Performance measured:
  - Encryption: 1.16 seconds
  - Analysis: 3.288853 seconds
  - Total: **4.448853** seconds
  - Test system specification
    - Intel i9-12900KF @ 3.19GHz
    - 32GB memory
    - Windows 11 Pro 64-bit
- No plaintext (or part of it) was written on the storage and/or memory during process.

Performance Comparison with Homomorphic Encryption



Calculation time

The time taken for each calculation is shown below.

- Generating a bootstrappable key: 37 secs (6.6 gb)
- Encrypting data: 6.27 secs (8 kb > 654.3 mb) / preprocessed
- Statistical analysis: 141.81 secs
  - Statistics computation

Calculation	Data	Execution time (Sec)
Delete column	Branch	1.08
Subtraction between columns	Asset-loan	6.34
Column mean	Loan	15.37
Standard deviation	Annual Salary	29.25

  - Categorical calculation

Calculation	Data	Execution time (Sec)
Average by category	Card spending - Female	12.00
Average by category	Card spending - Male	9.63
Average by category	Card spending - 20s	13.62
Average by category	Card spending - 30s	13.51
Average by category	Card spending - 40s	13.86
Average by category	Card spending - 50s	13.45
Average by category	Card spending - 60s	13.70

- Machine learning (logistic regression): 89 secs

Source: https://guide.ncloud-docs.com/docs/en/hha-example

	Company 'C'	Vesper 1.2c
Bootstrap Key Generation	37 seconds	0
Encrypting Data	6.27 seconds	0 (unmeasurable)
Analysis	141.81 seconds	0.022377 seconds
Total	**185.08** seconds	**≤ 0.1** seconds

- Vesper does not require any plaintext preprocessing before the calculation.
- Vesper only used 40.7KB of space.
  - It includes the Vesper encryption key (1MB).
  - Company 'C' product used (at least) 6.6GB of space.

- Notes
  - Measure for the machine learning was not performed for Vesper.
  - The resource and calculation time data are from 'NAVER Cloud Platform' and may not contain the up-to-date information.

Unreal Security

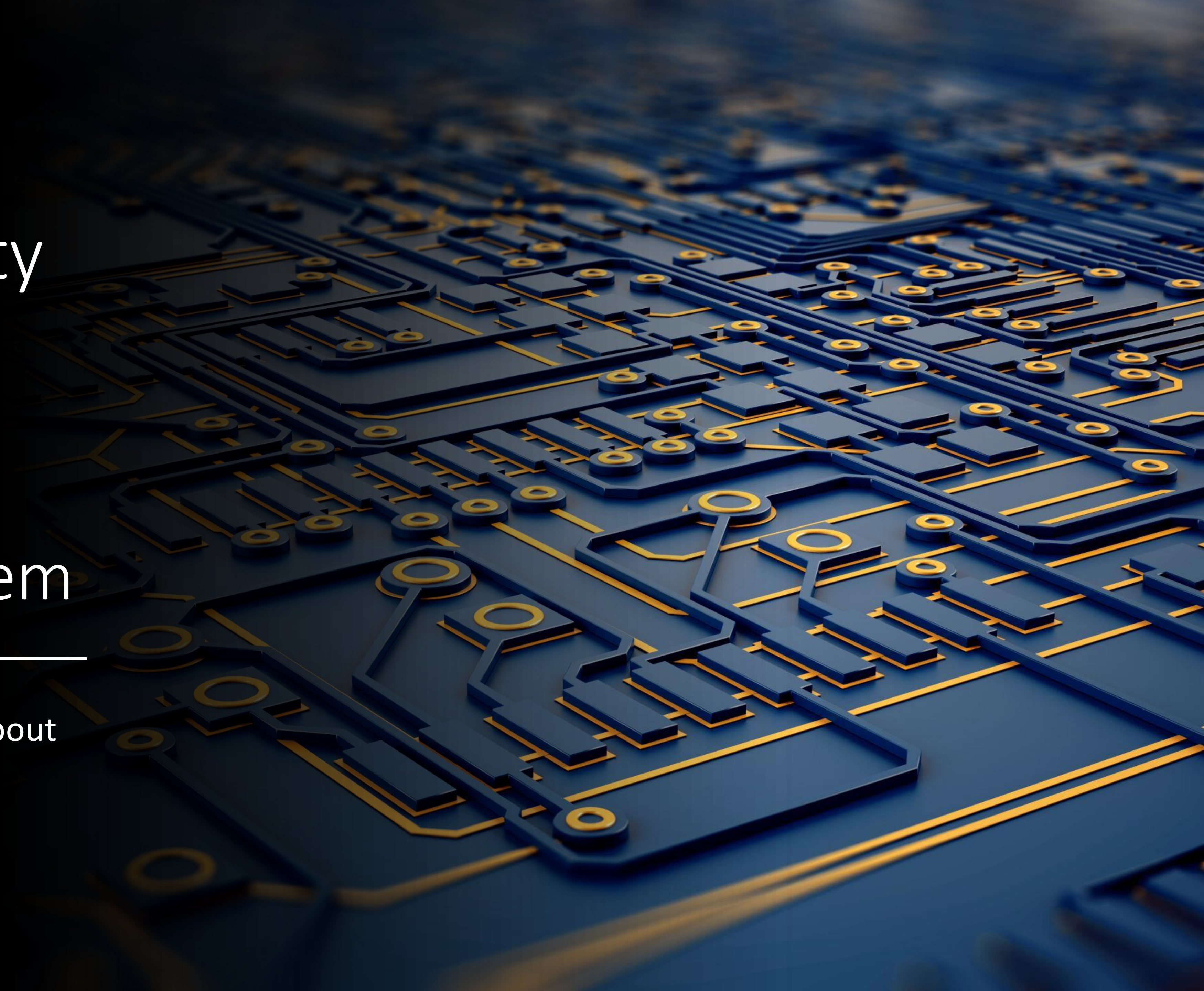
for

Embedded System

Solution for those who worry about

'Anti-temper'

on the system level



Security of Executables and Current Solution

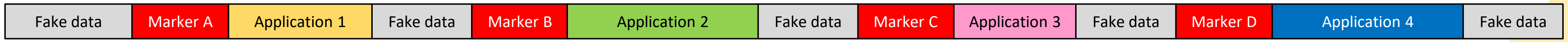
- Only the data (stored and/or transmitted) are encrypted but not the executable 'application.'
    - Most security requirements are for the static data.
    - Application software contains 'how to handle' information about proprietary or even classified system devices.

- High-level security applications require multiple level of security barriers (even physical destruction).
    - 'Leaked' software may be analyzed by competitors/adversaries → may expose proprietary information and technology.
    - Regardless of the sophisticated authorization method to 'unlock' the obfuscation (hiding), the codes are there.

- Software can be loaded encrypted and can be decrypted before use. However, the problem is that:
    - There is no way to check and verify the final 'loaded' software is 'not polluted' while encrypted.
        - Software verification process, such as hash check, requires the software to be <u>decrypted and stored</u> (in memory) before execution.
    - It is impossible to delete/change the loaded software after release even if the authorization data has been compromised.
        - Once the key (passwords) needs to be changed or has been compromised, the prepared encrypted software set becomes useless. The entire certification processes must be done again even when there is no single line of code change.
    - Embedded system generally does not have extra resources (memory/storage and processor power) to handle encryption.

Example Scenario 1: The latest and greatest AI car

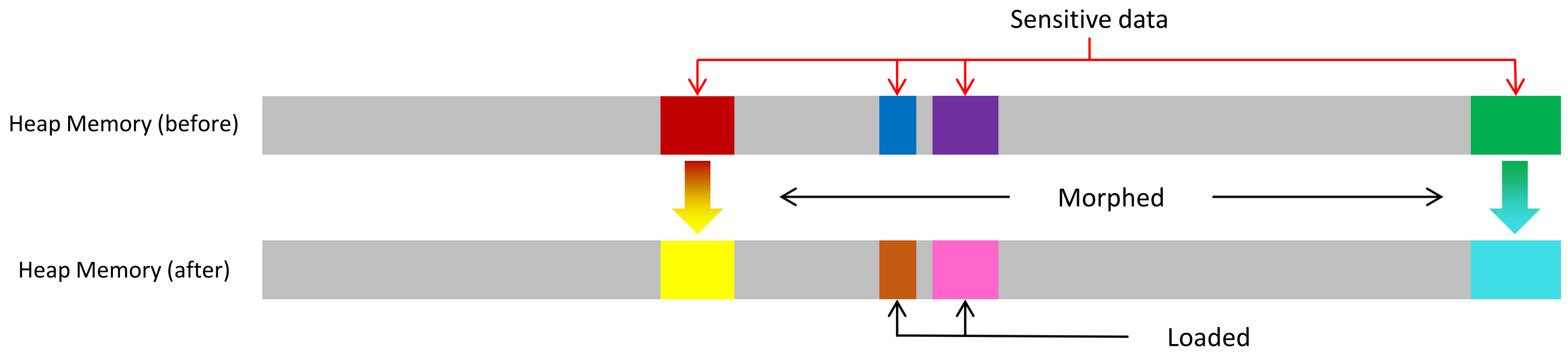
Vesper-encrypted data loaded onto the memory of an embedded system



(* Sizes are not in any meaningful scale.)

- Situation
  - A highly secure embedded system (for this AI car) powers up, tests and initializes system, starts system services, loads OS, and system software checks and identifies the equipped devices.
  - The cars of this model can be equipped with wide range of sensors/devices, from generic to the top-of-the-line ones that no other competitor can release with their cars yet. Thus, the control mechanism for the device and the information regarding data and timing specs are highly proprietary (or even classified).
  - System needs to load the application(s) based on the equipped devices. Each application handles the specific device and may even contain some AI algorithms specific for the device.
- Vesper can:
  - Search the target marker for the device directly from the encrypted data block so that proper application can be loaded.
  - Request the verification hash info from the authentication server to verify the application is ‘clean’ as released.
  - When the car powers down, during the shutdown process, Vesper can key-morph the data to invalidate the ‘old’ key and/or to prevent hackers from intrusion during sleep. The new key will not be stored locally (in the car) and can only be downloaded from the server once the owner gets authenticated.

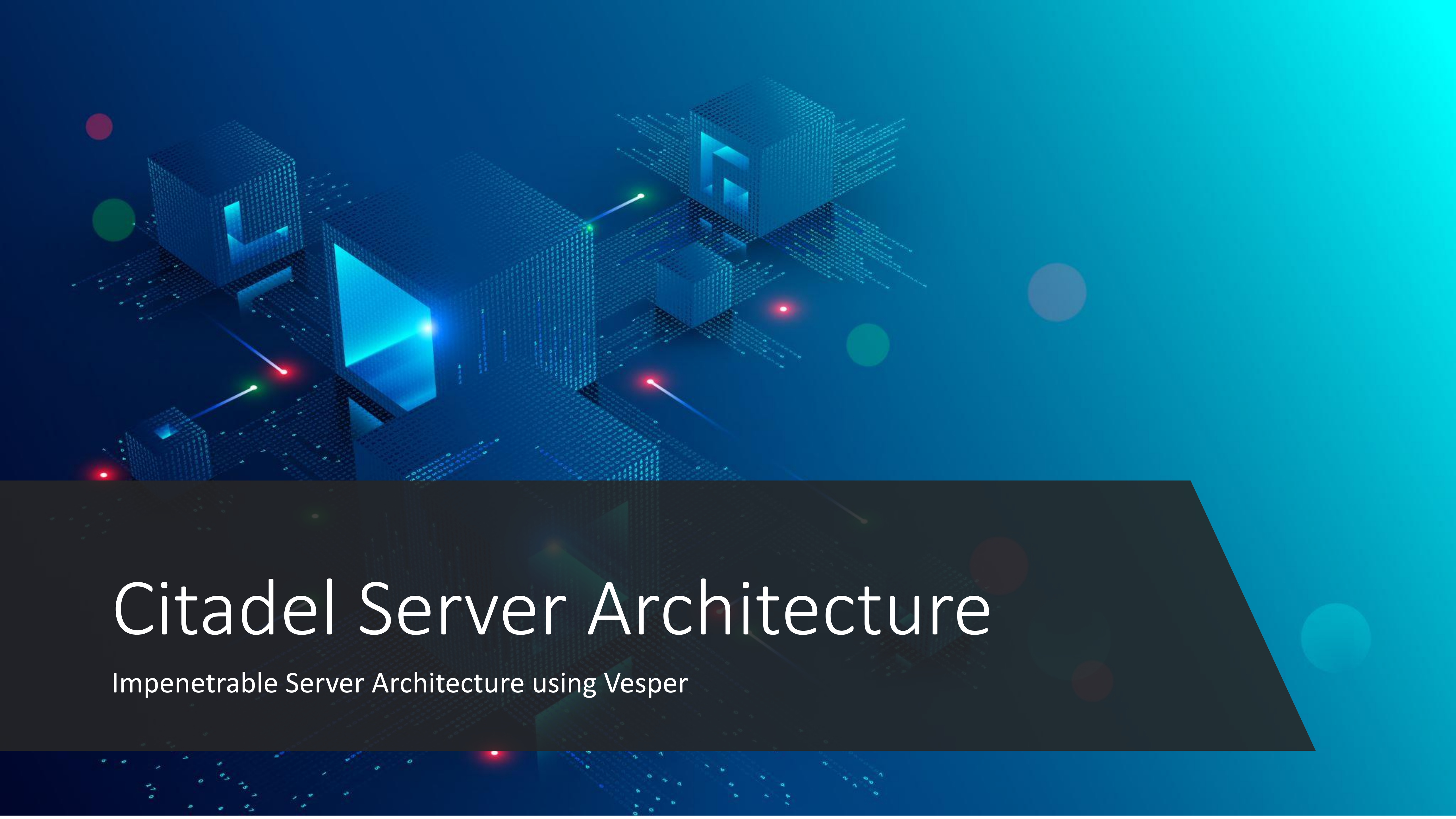
Example Scenario 2: Real-time Update of Security Data



- Situation
  - Application itself is not sensitive or proprietary, but the data parts are. (personal data, location info., filter values, etc.)
  - Most embedded systems have data mapped into the specific memory location.
  - As situation changes, some crucial data parts must be updated/changed. (Such as target change, timeline update, etc.)
  - Just as a routine security procedure, some encrypted data in the memory needs to be re-encrypted with updated keys.
- Vesper can:
  - Transmit/receive encrypted data and override parts of the data dynamically
  - Utilize obfuscated (fake) regions in the update data to enhance the security (possible interception during transmission)
  - Transform the encrypted data with the new key using key-morph
    - Vesper does NOT need an additional resource (memory) to convert/morph encrypted data.

Example Scenario 3: Secure Boot

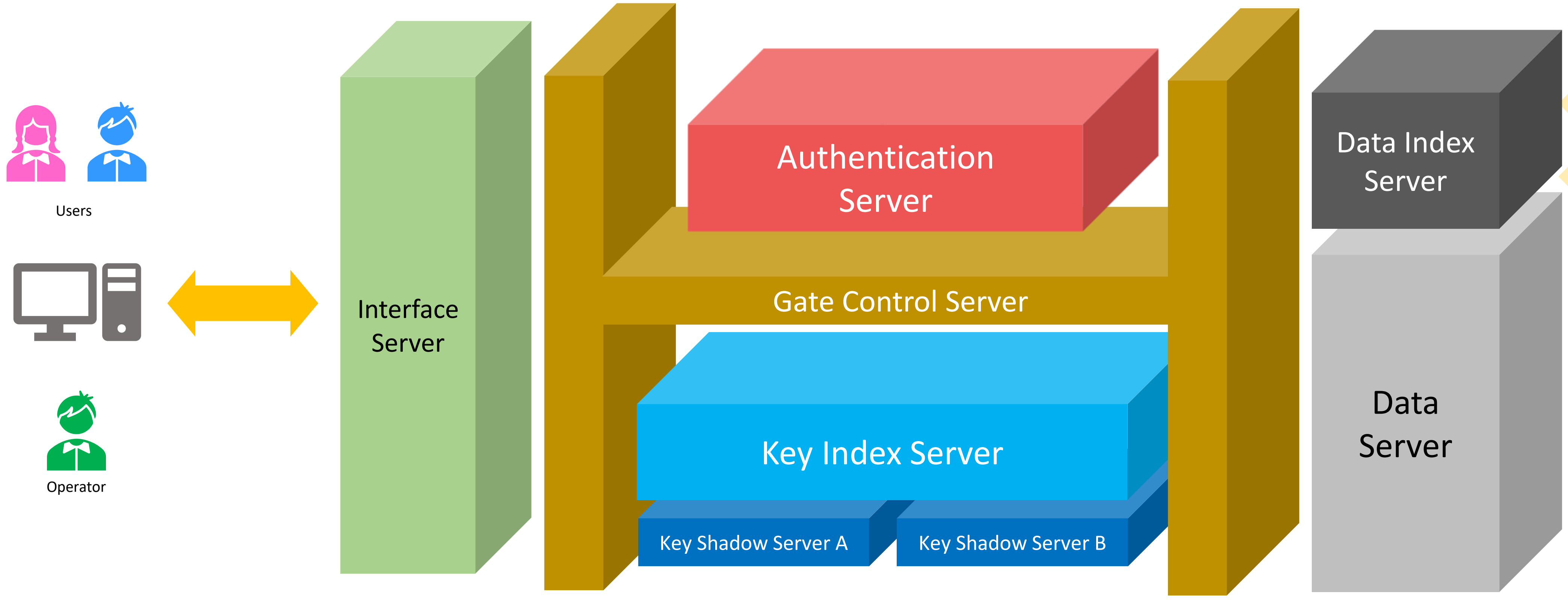
- Situation
  - Embedded system is most vulnerable while powered down.
  - Upon power up, bootloader cannot easily verify the authenticity of the system application.
    - Checksum and/or hash checking are static and must be updated/changed every time when the application get updated/changed.
      - If the intruder is knowledgeable enough to replace the application, he/she can easily bypass the checksum/hash checking.
    - Verification using encryption can be costly and may require additional resources (memory).
      - The static decryption program can easily be fooled if the original key is replaced with the hacked one for the hacked application.
- Vesper can:
  - Key-morph the entire application before shutdown and hide the randomized key within the fake block.
    - Encrypted application and its key change every time the system gets turned off.
  - Perform key-morphing of the entire application periodically to invalidate current key and replace with the new one.
  - Decrypt the entire application upon power up after key verification.
    - Vesper is very light-weight and does not require ‘unexpected’ additional resource.
    - Periodic key-morphing can be performed as a background service while the system is running.



Citadel Server Architecture

Impenetrable Server Architecture using Vesper

Secure Server Architecture – ‘Citadel’



- All servers within the system are independent from each other.
  - All servers must go through the authentication process before communicating, and ‘authentication server’ verifies and grants access privileges.
- Gate control server monitors and controls all traffics between servers.
- Connectors and other small helper objects are not shown.

Responsibilities of Servers in Detail

- Interface Server (a.k.a. Frontend)
    - Only gate between external and internal worlds
- Gate Control Server
    - Monitor and control all internal and external communication between users and servers
    - Verify communication using checksum or MAC (message authentication code) to prevent intrusion
- Authentication Server
    - Verify and grant/deny accessibility of users and servers to access specific resource and/or data
    - Servers cannot directly access other servers and/or their components even if they are parts of the Citadel system.
- Key Management Server (Key Server)
    - Key Index Server keeps track of all the list of the keys
    - Key Shadow Servers consist of two (or more) servers that keep the XOR masked key data.
        - All masked key data must be XORed to retrieve the real key.
- Data Index Server
    - Keep the list of the data and their key codes stored in the Data Server
- Data Server
    - Store all data encrypted
    - Keys for the stored data are stored and managed by Key Server.

Security of Citadel Server System

- No single-point failure
  - Most, if not all, current popular servers (database, web server, etc.) work as a ‘single’ entity of its own.
    - Once intruded, all data are available for grab.
- All data are encrypted – No data is stored unencrypted
  - Leaked/stolen data is effectively useless without proper key.
  - Keys and encrypted data are stored separately.
- Keys are masked and stored distributed over multiple servers.
  - Unless hackers take the entire ‘physical’ Citadel servers, it is impossible to unmask key
    - Finding the right key is another problem. Even system operators cannot find the right key for a data.
- Vesper provides Key-morphing feature.
  - Once intrusion and/or unauthorized access are found, Vesper can invalidate selected (or even entire) keys instantly.
- The goal of the Citadel architecture is to make it *impossible to manually search and retrieve data*.

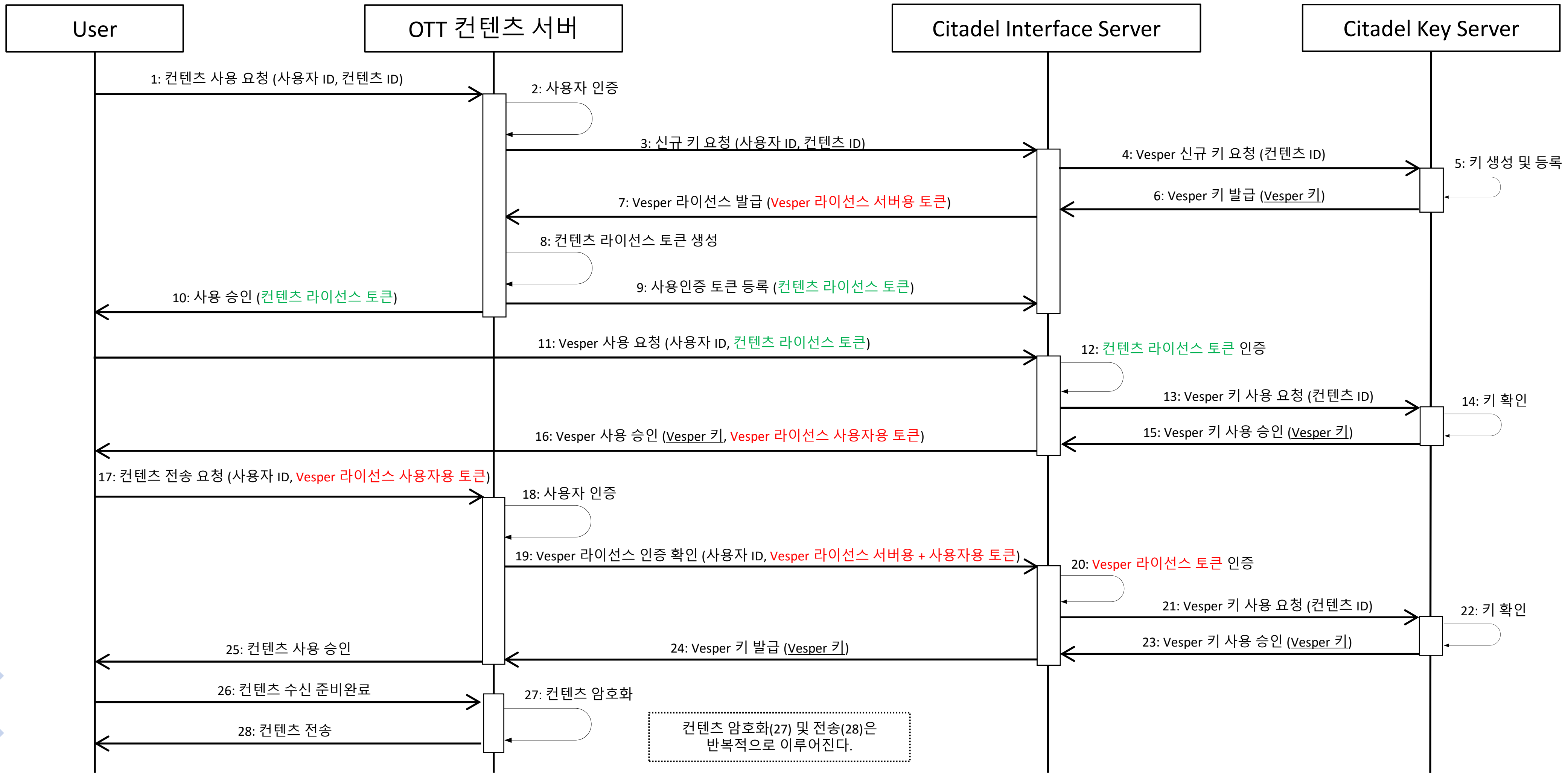
Media DRM Server System

Securing Digital Rights for Multimedia Contents

DRM (Digital Rights Management) Using Citadel

- Streaming Key Management
  - Vesper is super fast; encryption and decryption can be done in real-time even for the 4K HD movies.
  - Different key will be assigned for individual customer for the same media.
- Combined with Digital Steganography
  - Special ‘hidden’ signature can be injected/embedded into the media – hidden signature can be used to trace the origin.
    - Each user will use the unique Vesper key, and it can be used to identify the source of the illegal distribution of the digital media.
  - Hidden code can be embedded to personalize and/or customize for individual customer.
    - Personal message for family members, friends, lover, etc.
    - Personalized greetings for customers (fans)
    - Hidden sensitive message for specific individuals or groups
  - Steganography can be used for image, music files, movies, and any other digital media formats.
  - Replacement for NFT (non-fungible token)
- Proof of Authenticity
  - Use with steganography and hash algorithms, such as SHA-256, the ownership and authenticity can be easily verified.

OTT Contents 서버와 Citadel 서버 간 연동의 예


User
OTT 컨텐츠 서버
Citadel Interface Server
Citadel Key Server
1: 컨텐츠 사용 요청 (사용자 ID, 컨텐츠 ID)
2: 사용자 인증
3: 신규 키 요청 (사용자 ID, 컨텐츠 ID)
4: Vesper 신규 키 요청 (컨텐츠 ID)
5: 키 생성 및 등록
6: Vesper 키 발급 (Vesper 키)
7: Vesper 라이선스 발급 (Vesper 라이선스 서버용 토큰)
8: 컨텐츠 라이선스 토큰 생성
9: 사용인증 토큰 등록 (컨텐츠 라이선스 토큰)
10: 사용 승인 (컨텐츠 라이선스 토큰)
11: Vesper 사용 요청 (사용자 ID, 컨텐츠 라이선스 토큰)
12: 컨텐츠 라이선스 토큰 인증
13: Vesper 키 사용 요청 (컨텐츠 ID)
14: 키 확인
15: Vesper 키 사용 승인 (Vesper 키)
16: Vesper 사용 승인 (Vesper 키, Vesper 라이선스 사용자용 토큰)
17: 컨텐츠 전송 요청 (사용자 ID, Vesper 라이선스 사용자용 토큰)
18: 사용자 인증
19: Vesper 라이선스 인증 확인 (사용자 ID, Vesper 라이선스 서버용 + 사용자용 토큰)
20: Vesper 라이선스 토큰 인증
21: Vesper 키 사용 요청 (컨텐츠 ID)
22: 키 확인
23: Vesper 키 사용 승인 (Vesper 키)
24: Vesper 키 발급 (Vesper 키)
25: 컨텐츠 사용 승인
26: 컨텐츠 수신 준비완료
27: 컨텐츠 암호화
28: 컨텐츠 전송
컨텐츠 암호화(27) 및 전송(28)은 반복적으로 이루어진다.