Searchable Encryption and CTI Sharing

1^{st} Summer School on Security and Privacy in 6G Networks

António Pinto
apinto@estg.ipp.pt

June 24 - 28, 2024
Faculty of Computer Science and Engineering
UCM, Madrid, Spain

Contents

Introduction

- Searchable encryption **enables remote search operations** to be performed **over encrypted data**, without the need to decrypt it
- May enable **cloud-based storage** of any type of data without compromising confidentiality or efficiency

Searching remotely encrypted data

- **Cloud**-based infrastructure and applications **combined** with the **GDPR** creates momentum for a greater adoption of the **encryption of data and logs**
- Remotely storing data or logs that might **contain personal information** should **use encryption**
- If remote data confidentiality is required, the most **common solution** is to use cryptography techniques to **encrypt all data before transferring** it to a remote cloud storage service
- Simplistic solution includes **transferring all data back**, **decrypting** it, and then **searching over the clear text**
- **Impractical** with data growth, does not make use of the **full potential** of cloud computing

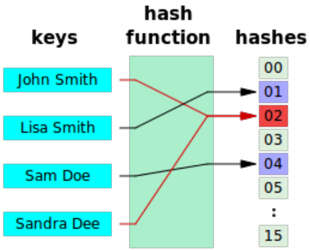
Contents

Hash function

Hash function

Function to quickly generate a fixed-size, pseudo-random block of output bits from a variable block of input bits.

- Often used for quick searches (*hash tables*)
- Enables fast duplicate detection



(source: Wikipedia, Jorge Stolfi)

Experimenting with hash functions

Exercise (15 minutes)

Obtain `hash.html` from the shared folder, open it with the browser and **create four single characters hashes** using any four values of **your choosing**, so that **at least two** results are the **same**.

How long did it take? **Why**?

Would take **more or less time** to obtain two equal hashes if you were using the **default output size**?

Secure hash function

Secure hash function

Hash function that must have some additional cryptographic properties. (R. Anderson, 2008)

Required properties

- *One-way*: For a given value y it must be computationally impossible to discover a x such that $H(x) = y$
- Resistant to weak collisions: For a given value x it must be computationally impossible to discover a x' such that $H(x') = H(x)$
- Collision resistant: It must be computationally impossible to discover a pair $x \neq y$ such that $H(x) = H(y)$

Secure hash function

Examples

- MD5 = Message Digest 5 [RFC 1321] - 32 bit operations
- SHA-1 = Updated SHA [NIST]
- SHA-2 = SHA-224, SHA-256, SHA-384, SHA-512 [NIST]
  - SHA-512 uses 64-bit operations

Secure hash function

Collisions

Two different datasets, however similar they may be, should yield different cryptographic digests.

Function	Input	Output
MD5	Security and Privacy	**6180**feadfbb5a5d93698b42458362bbe
MD5	**s**ecurity and **p**rivacy	**df09**5202cf8cb4979ca494853e979f8b

When two different data sets generate the same cryptographic digest, it says that we are facing a **collision**, meaning, the algorithm has been broken.

Secure hash function

Collision MD5

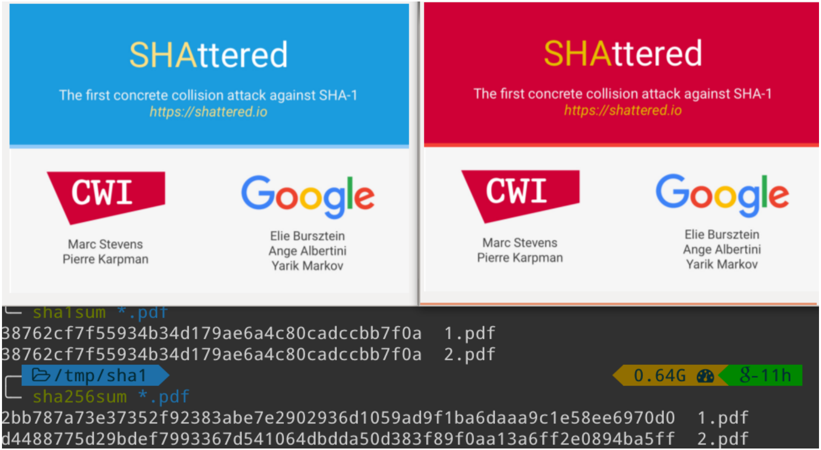
These images generate an MD5 collision

`http://natmchugh.blogspot.co.uk/2014/11/three-way-md5-collision.html`

Secure hash functions

Collision SHA1

These PDFs generate a SHA1 collision.



+info: `https://shattered.io/`

Experimenting with hash collisions

Exercise (30 minutes)

Obtain `collisions.zip` from the shared folder, unzip it and **calculate** the **MD5**, and **SHA1** hash values of all files.

Are there **any collisions**? Which ones? Are these algorithms secure?

– – Help – –
In Linux:

```
sha256sum *
md5sum *
```

In Windows command line (one file at a time):

```
certutil -hashfile fich.txt md5
certutil -hashfile fich.txt sha256
```

Contents

Encryption algorithms (or ciphers)

Quick recap!

- Modern cryptographic algorithms are based on **mathematical** calculations
- Such calculations must be **fast** to verify when having **complete** information
- (very) **Slow** when **part** of the information is **unknown**
- Some (asymmetric) ciphers rely on the **product** of prime numbers for these calculations [1]
  - Efficiently **factorizing large number** into **prime** factors is **computationally intensive** and considered a **hard problem**

[1] RSA public key is derived from the product of two large prime numbers

Factorization into prime numbers

- **Factors** are the **numbers** that allow for the **integer division** of **another number**
- Factors of the number 30 are: 30, 15, 10, 6, 5, 3, 2, and 1
- Limiting factors to **prime numbers**, we get **prime factorization**
- Prime factorization of 30:

$$2 \times 3 \times 5 = 30$$

Factorization into prime numbers

Exercise (15 minutes)

Factorize the **number 253 into prime** numbers. You may use software like calculators and spreadsheets.

Now, obtain `factorize.html` from the shared folder, and use it to factorize the **number 253 into prime** numbers. Take a careful look at the list of required iterations shown.

Encryption types

There are basically two types of ciphers:

- **Symmetric** encryption
  - Uses **one key for all**
  - Also known as **secret key encryption**
- **Asymmetric encryption**
  - Uses **two keys per participant**
  - Also known as **public key encryption**

Symmetric Encryption

- Uses a **single** secret key for **both encryption** and **decryption**
- **Fast and efficient** for large amounts of data
- **Suited** for scenarios where **parties already share a secret** key
- Enables **confidentiality**, but only **while** the secret **key is secure**
- Example: AES (Advanced Encryption Standard)

Experimenting with AES

Exercise (15 minutes)

Obtain `aes.html` from the shared folder and **experiment encrypting** and **decrypting** text.

Afterwards, obtain `ciphertext.txt` from the shared folder and **decrypt it**.

Candidate passwords: Brussels, Vienna, Sofia, Prague, Copenhagen, Berlin, Tallinn, Helsinki, Paris, Athens, Budapest, Dublin, Rome, Riga, Vilnius, Luxembourg City, Valletta, Amsterdam, Warsaw, Lisbon, Bucharest, Bratislava, Ljubljana, Madrid, Stockholm, Nicosia ...

Asymmetric Encryption

- Uses **pairs of keys** for encryption and decryption (**public and private** key)
- **Public** can be **freely distributed**, **private** must be **kept secret**
- Allows **secure communication without** prior **key exchange**
- **More complex** and uses **larger keys** then symmetric encryption
- Examples: RSA (Rivest-Shamir-Adleman), ECC (Elliptic Curve Cryptography), DSA (Digital Signature Algorithm)

Experimenting with RSA

Exercise (10 + 20 minutes)

Obtain `rsa.html` from the shared folder and **experiment encrypting** and **decrypting** text.

Afterwards, obtain `ciphertext2.txt` from the shared folder. Access `https://gchq.github.io/CyberChef` and copy the file contents to the *input* section of *CyberChef*.

Add the ”**From Base64**” and ”**RSA Decrypt**” operations, in this order, to *CyberChef*. **Fill** in the **private key** (file `privatekey.txt`), resulting in the text decryption.

Contents

Searchable Encryption (SE)

- In **clear text search** operations, **entity sends keywords** to the server and **retrieves matching data**
- Knowledge of both **keywords and matching data** becomes **known to the server**
- SE is a technique that **preserves confidentially** while enabling **server-side searching** [1]

SE consists in a cryptographic method that encrypts data by such a scheme that enables remote keyword searches to be conducted over the encrypted data.

Main challenges

- There is **no one-fits-all** solution; over the years, various solutions have been proposed
- In some, researchers focused on **efficiency**, others focused on the **security and privacy**, others on the **expressiveness** of queries [2]
- Any searchable encryption mechanism must strike a **balance** between 3 issues: **security, efficiency** and **expressiveness** (or searchability)

- Depending on the **application scenario**, the **importance** of each factor **may vary**

Existing schemes

- Direct
  - Search operations are performed **directly** and sequentially **over the ciphertext**
  - **Search time is linear** to the **size of the data** stored on the server
- Index-based
  - Use encrypted **index of documents or keywords**
  - Can **increase search performance** because queries are performed **using a trapdoor** function to create **search tokens** (or trapdoors)
  - Requires **higher computations** at the **storage** phase in order to **populate the index**

Forward vs. Inverted Indexes

Comparison [3]

document	keyword
1	k1, k3, k5
2	k2,k4
...	...
n	k

Figure: Forward index

keyword	document
k1	1,3,5
k2	2,4
...	...
k	n

Figure: Inverted index

- Forward index results in search time of $O(n)$, where n = number of documents
- Inverted index results in search time of $O(|D(k)|)$, where $|D(k)|$ = number of documents containing the keyword k

Experimenting with indexes

Exercise (10 minutes)

Obtain `forwardindex.html` and `reverseindex.html` from the shared folder and open them with your browser.

Afterwards, **select** any **keyword** and **search** for it in **both indexes**. Take **note** of the **number of iterations** took by each search. Analyse the results.

Trapdoor

- Functions used to create **search tokens**, which **identify** the presence of a **given term** in the index
- **Receive** a plaintext value (**term**) and a **key**, **producing** a **ciphertext value** that corresponds to the term
- Analogous to *hash* functions $\rightarrow$ Example HMAC (Hash-based Message Authentication Code)

Trapdoor, experimenting with HMAC

Exercise (20 minutes)

Obtain `tradoor.html` from the shared folder, open it with the browser and create four trapdoors using the following values:

Text	Key
1.1.1.1	VeryStrongPassword
atacker.machine.online	atacker.machine.online
1.1.1.1	AnotherPassword
atacker.machine.online	AnotherPassword

Analyse the obtained results. Are **results** for ”1.1.1.1” the **same or different**? Why?

Change the underlying secure hash algorithm to **SHA-512**, by **editing the source** code. What changed in the **results**? Why?

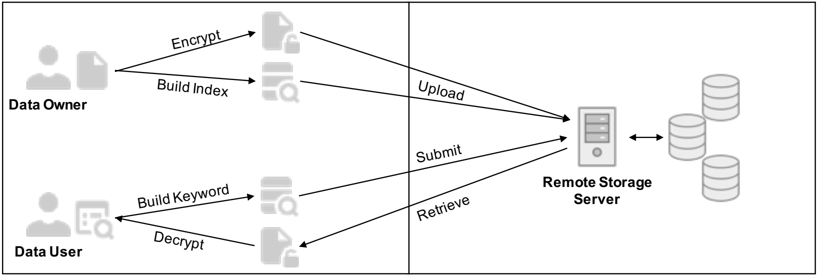
Inverted index in searchable encryption

~~keyword~~	trapdoor	encrypted data
~~k1~~	t1	Doc1, Doc3, Doc5
~~k2~~	t2	Doc2, Doc4
…		…
~~kn~~	tn	DocN

Uses **trapdoors** for **encrypted data** indexing

Searchable encryption system

System model [4]



- Data owner has sensitive information to be remotely stored (including trapdoors)
- Data user is authorised to search the remote data (using trapdoors)

Classes of searchable encryption

- **Symmetric Searchable Encryption** (SSE), based on symmetric encryption algorithms
- **Public key Encryption with Keyword Search** (PEKS), based on asymmetric encryption algorithms

Symmetric Searchable Encryption

Selected solutions

- In Song's [5] (2000), **each word** is **encrypted separately** and **concatenated with a special hash** value; in search operations, the server can extract the hash value and test if the value matches (low search efficiency)
- In Goh's [6] (2003), an **index per document** is used; the index is created using **bloom filters**[2][7]**.** In practice, returns that an element is either definitely not in the set or possibly in (false positives)
- In Chang's [8] (2005), a **two-index scheme** was used; allowing for the remote storage of the second index (encrypted)
- In Curtmola's [9] (2011), a **reverse index** was used; searching is more efficient but index modifications are costly

---

[2] Test if an element is present on a set.

Public key Encryption with Keyword Search

Selected solution

- PEKS using Identity Based Encryption (IBE) [10] was proposed by Boneh [11] (2004)
  - IBE derives **public keys** from each entity's identity
  - Entities **connect to Private Key Generator** (PKG) to request private keys of public keys
  - PKG **computes master** public and private key pair **per entity**
- In PEKS, keywords act as the identity
- Sender produces ciphertexts with the receiver's public key
- Only private key owners can generate search trapdoors

Public key Encryption with Keyword Search

Selected solution (more detail)

- Data storage
  - **Clear text** encrypted by **standard public key** system
  - Each **keyword** is encrypted with **IBE** scheme
  - **Concatenation** of both ciphertexts is **sent to server**
- Searching
  - Master private key (in PKG) used to **derive private key** for **keyword to be searched**
  - **Key** used in the **trapdoor** verification function
  - Server will try to **decrypt all** the existing ciphertexts
  - If decryption is **successful** the **keyword is present**

Public key Encryption with Keyword Search

Selected solution (some more detail)

- Disadvantages
  - Trapdoors are produced by a **deterministic** encryption system (**same keyword, same trapdoor**)
  - **Server** can test **old trapdoors on future** documents
  - **Requires secure channel** with PKG
  - **Query efficiency** drastically **reduces** with the **increase of stored** data

Query expressiveness

- **Research** has been conducted to provide **more than single keyword** queries
- **Server-side multi-keyword conjugation** was introduced by Golle in 2004 [12], others followed
- **Fuzzy keyword search**, tolerating **minor typos** and formatting inconsistencies, was proposed by Park in 2007 using **char-by-char encryption** [13], others followed
- **Ranked keyword search** returns most relevant documents first, was proposed by Wang in 2010 [14, 15] using **order-preserving deterministic** encryption, others followed
- **Expressiveness is a trade-off** achieved at expense of efficiency or security

Contents

Sharing of Classified Threat Intelligence (CTI)

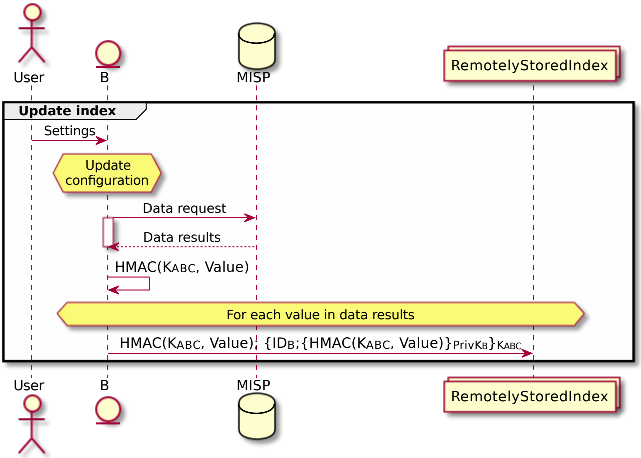
- **CTI** platforms are important **against cyberattacks**
- **MISP** enables **dissemination** of threat information within a **community**
- MISP **assumes trust** within the community and **does not encrypt** exchanged information
- **Not suited** for **classified** information exchange between **military organizations**
- In [16, 17] we proposed the **use of searchable encryption** to impose greater control over information sharing in MISP

CTI Sharing with Searchable Encryption

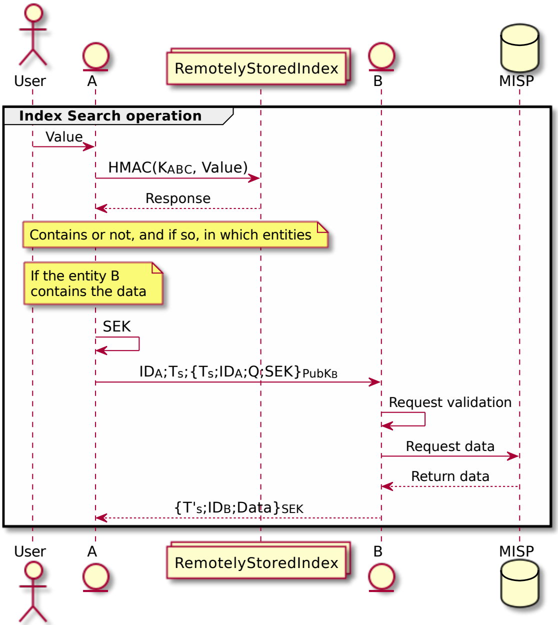
Key benefits

- CTI is **always encrypted** (at **rest**, in **transit** and while **used**)
- It is also **privacy-friendly** (only the ones with the IoC, can generate the corresponding trapdoor)
- Can be **remotely stored**, with **less worries** regarding its confidentiality
- **Indexes** can be **replicated and distributed**, improving performance, responsiveness, load balancing and resilience

Sharing of Classified Threat Intelligence - **Indexing**



Sharing of Classified Threat Intelligence - **Searching**



Contents

Looking forward

1. Availability and cloud independence

- An **index is required** in order to **store trapdoors**
- **Index** is frequently **stored remotely**, as a service, on a cloud provider
  - **Single point of failure** or **vendor lock-in**
  - Issues like **index replication**, **index distribution**, **index synchronization** will lead to new research (**PRIVATEER**)
- Extend the use of trapdoors (**regular and reference trapdoors**) is another possibility
- Searchable encryption **service requiring only partially trust** from **users** is yet another possibility

Looking forward

2. Machine learning techniques

- **Machine learning** is frequently proposed as an **attack vector**
  - **Cloud** provider storing the index may **correlate information** and form some **knowledge** about data
- **Assess** use of machine learning **to attack the proposed** solutions
- If possible, research way of **hampering its use** introducing **dummy data** (and/or dummy requests)
  - How much? How often?

Searchable Encryption and CTI Sharing

1st Summer School on Security and Privacy in 6G Networks

António Pinto
apinto@estg.ipp.pt

June 24 - 28, 2024
Faculty of Computer Science and Engineering
UCM, Madrid, Spain

References

[1] Yunling Wang, Jianfeng Wang, and Xiaofeng Chen.
Secure searchable encryption: a survey.
Journal of Communications and Information Networks, 1(4):52–65, 2016.

[2] Rui Zhang, Rui Xue, and Ling Liu.
Searchable encryption for healthcare clouds: a survey.
IEEE Transactions on Services Computing, 11(6):978–996, 2017.

[3] Christoph Bösch, Pieter Hartel, Willem Jonker, and Andreas Peter.
A survey of provably secure searchable encryption.
ACM Computing Surveys (CSUR), 47(2):18, 2015.

[4] Jianjun Zhang.
Semantic-based searchable encryption in cloud: issues and challenges.
In *2015 First International Conference on Computational Intelligence Theory, Systems and Applications (CCITSA)*, pages 163–165. IEEE, 2015.

[5] Dawn Xiaoding Song, David Wagner, and Adrian Perrig.
Practical techniques for searches on encrypted data.
In *Security and Privacy, 2000. S&P 2000. Proceedings. 2000 IEEE Symposium on*, pages 44–55. IEEE, 2000.

[6] Eu-Jin Goh et al.
Secure indexes.
IACR Cryptology ePrint Archive, 2003:216, 2003.

[7] Burton H Bloom.
Space/time trade-offs in hash coding with allowable errors.
Communications of the ACM, 13(7):422–426, 1970.

[8] Yan-Cheng Chang and Michael Mitzenmacher.
Privacy preserving keyword searches on remote encrypted data.
In *International Conference on Applied Cryptography and Network Security*, pages 442–455. Springer, 2005.

[9] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky.
Searchable symmetric encryption: improved definitions and efficient constructions.
Journal of Computer Security, 19(5):895–934, 2011.

[10] Dan Boneh and Matt Franklin.
Identity-based encryption from the weil pairing.
In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[11] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano.
Public key encryption with keyword search.
In *International conference on the theory and applications of cryptographic techniques*, pages 506–522. Springer, 2004.

[12] Philippe Golle, Jessica Staddon, and Brent Waters.
Secure conjunctive keyword search over encrypted data.
In *International Conference on Applied Cryptography and Network Security*, pages 31–45. Springer, 2004.

[13] Hyun-A Park, Bum Han Kim, Dong Hoon Lee, Yon Dohn Chung, and Justin Zhan.
Secure similarity search.
In *2007 IEEE International Conference on Granular Computing (GRC 2007)*, pages 598–598. IEEE, 2007.

[14] Cong Wang, Ning Cao, Jin Li, Kui Ren, and Wenjing Lou.
Secure ranked keyword search over encrypted cloud data.
In *2010 IEEE 30th international conference on distributed computing systems*, pages 253–262. IEEE, 2010.

[15] Cong Wang, Ning Cao, Kui Ren, and Wenjing Lou.
Enabling secure and efficient ranked keyword search over outsourced cloud data.
IEEE Transactions on parallel and distributed systems, 23(8):1467–1479, 2011.

[16] Ricardo Fernandes, Pedro Pinto, and António Pinto.
Controlled and secure sharing of classified threat intelligence between multiple entities.
In *2021 IEEE International Mediterranean Conference on Communications and Networking (MeditCom)*, pages 525–530, 2021.

[17] Ricardo Fernandes, Sylwia Bugla, Pedro Pinto, and António Pinto.
On the performance of secure sharing of classified threat intelligence between multiple entities.
Sensors, 23(2):914, Jan 2023.