Performance Evaluation of Searchable Symmetric Encryption in Wireless Sensor Networks

Cristina Muñoz(✉), Lucas Rocci, Eduardo Solana, and Pierre Leone

Computer Science Department, University of Geneva, Carouge, Switzerland
{Cristina.Munoz,Eduardo.Solana,Pierre.Leone}@unige.ch,
Lucas.Rocci@etu.unige.ch

Abstract. The distributed nature of Wireless Sensor Networks leads to the use of cloud databases that need to be protected when dealing with sensitive content. In this context, Searchable Symmetric Encryption provides the appropriate framework to perform secure searches. This work proposes a combination of secure indexes with Bloom Filters to efficiently address searches in encrypted content. We evaluate the performance of two different strategies to populate Bloom Filters in XM1000, Z1 and TelosB wireless sensor devices: (1) we first consider four cryptographic hash functions using the double hashing technique and truncating message digests; (2) we then select five symmetric encryption algorithms and two fast hash functions also with double hashing. We conclude that the best strategy for securing indexes is AES plus a fast FNV hash function and double hashing.

Keywords: Searchable Symmetric Encryption · Wireless Sensor Networks · Bloom Filters · Cloud Storage

1 Introduction

The ubiquitous model of the Internet of Things (IoT) leads to manage information through secure cloud systems. In the Database as a Service (DBaaS) model data is stored and managed in the cloud. This model assumes that documents are securely stored and only accessible to authorized users. In this process the database provider is usually considered as an "honest but curious" adversary with respect to the documents stored. For example, one can imagine a DBaaS for a Wireless Sensor Network (WSN) used in a Smart City that manages sensitive citizens data such as physical characteristics, location, actions, etc. In this scenario, only public security forces should access this private information.

Searchable Symmetric Encryption (SSE) offers a solution to search for specific encrypted documents on a database. In this context, the main challenge is to prevent the database provider from extracting relevant information related to the search process.

Secure indexes based on Bloom Filters (BFs) [4,10,14,17,19,20] are generally chosen because of their efficiency. Since our architecture relies on performance constrained devices, we have considered that BFs constitute the ideal candidate.

B. Mandler et al. (Eds.) IoT 360° 2015, Part I, LNICST 169, pp. 40–51, 2016.
DOI: 10.1007/978-3-319-47063-4_4

A BF [18] is defined as a probabilistic data structure that efficiently manages membership of a certain number of elements. Secure indexes based on BFs use cryptographic hash functions to add elements to the filter.

The aim of this research is to improve the construction of indexes in terms of security and performance. We focus on the performance evaluation of different strategies to securely populate BFs using well-known cryptographic algorithms. This means that the results of our research are valid for all methods that use secure indexes and trapdoors based on BFs in a WSN.

We compare the performance of mote devices in terms of ROM, RAM, energy and execution time using two different strategies:

- First, we evaluate four widely used cryptographic hash functions that are directly applied to keywords in order to fill in BFs in a secure way.
- The second strategy relies on encryption algorithms and fast non-cryptographic hash functions. Keywords are first encrypted and then hashed in order to populate BFs in an efficient manner.

Finally, we discuss the suitability of the aforementioned algorithms and their advantages and disadvantages in terms of security.

The rest of this paper is organized as follows: Sect. 2 points out the related work on the field. Section 3 details our methodology. Section 4 presents the results obtained. Finally, Sect. 5 summarizes our work.

2 Related Work

The first research that presents secure indexes is based on the use of BFs [4]. HMAC-SHA1 is proposed as keyed hash function but its performance is not discussed. According to [17] this scheme is still the most secure for full searches.

Secure indexes are usually based on BFs due to their efficiency [4,10,14,17,19,20]. In [17] a strategy to support searches that allow comparisons character by character is presented. In [10], secure indexes based on BFs are introduced to secure the deduplication of data in which duplicated encrypted files are removed to improve the memory size of the cloud database. In [19] a multi-keyword fuzzy search technique is presented. It uses a locality-sensitive hashing technique to support the misspelling of keywords. Similarity searches based on a symbol-based trie-traverse technique have been proposed in [20]. Moreover, in [14] a secure anonymous database search is presented by adding a query router between data searchers and index servers. Its purpose is to enforce an authorization before accessing secure indexes based on BFs. In this process, the query router is not allowed to gain data about the queries and their results.

Furthermore, there exists some research concerning the use of secure BFs in WSNs. In [11] a technique for encrypted data aggregation is used where secure BFs reduce transmission costs. Besides this, a method to support in-network processing [22] while securing traffic has also been developed in WSNs.

Finally, it is remarkable to mention that lightweight block ciphers have been evaluated in sensor motes. In [3], AES and XTEA are highlighted due to their good performance.

3 Methodology

In this section we focus on our proposed methodology to implement SSE in WSNs. First of all, we describe the scenario and the required processes for saving and searching an encrypted document in a cloud database. Then, we detail how to fill in secure indexes based on BFs using: (1) cryptographic hash functions and (2) an encryption algorithm plus a fast hash function.

3.1 Scenario

A SSE encryption scenario requires three agents: (1) the Encrypted Database (ED), (2) the Data Owner (DO) and (3) the Data Searcher (DS). In this paper, we consider that agents are attached to a WSN composed of constrained devices. This means that efficiency is essential in the communication process. Figure 1 shows the communication process to generate and retrieve data using SSE in a WSN:

1. The first requirement is that DOs and DSs participate in a key exchange protocol to share the same symmetric key.
2. The DO generates a document to store on the ED.
3. The document is symmetrically encrypted and an index containing relevant keywords is generated to facilitate searches. In our case, the index is a BF associated to the document. To prevent information leakage, index entries should also be encrypted with a pseudo-random function as described in [4]. We then insert random 1's in order to hide the number of keywords in the index. Finally, the associated secure index and the encrypted document are stored in the ED.
4. An authorized user holding the proper symmetric key generates a trapdoor to retrieve documents associated to certain keywords. Then the secure trapdoor is sent to the ED.
5. The ED searches for indexes that match the trapdoors and sends the associated encrypted documents to the DS.
6. The search results may include false positives but this does not constitute a security breach since the DS only decrypts documents protected with his key.

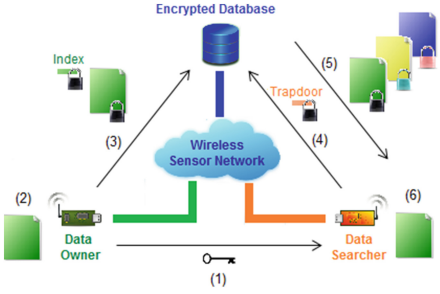


Fig. 1. SSE in Wireless Sensor Networks.

3.2 Cryptographic Hash Functions

For the evaluation of cryptographic hash functions we have chosen four widely used algorithms:

MD5 (1991). MD5 is based on a Merkle-Damgård function and produces 128 bit digests [15]. Each message block implements 4 rounds of 16 operations each one. This cryptographic hash function is highly vulnerable to collision attacks so that it was substituted by SHA1. Pre-image attacks are theoretically possible but are not practical due to their high computational cost. Nowadays, it is used as a checksum hash to verify the integrity of a file.

SHA1 (1995). SHA1 relies on a Merkle-Damgård function and produces 160 bit digests using 80 rounds [16]. It is vulnerable to collision attacks for high computational efforts. For this reason, it was decided to substitute it for SHA2 that presents no vulnerabilities.

SHA2 (2001). SHA2 is also based on a Merkle-Damgård function and produces 224, 256, 384 or 512 bit digests [16]. Digests of 224 and 256 can use 64 or 80 rounds while the rest need 80 rounds. Pre-image and collision resistance have been compromised for a limited number of rounds but it is still considered secure.

SHA3 (2012). SHA3 depends on a Sponge construction and produces 224, 256, 384 or 512 bit digests using 24 rounds [13]. SHA3 is a very recent algorithm with no significant flaws so far.

Additional Methods Used. Besides the cryptographic hash functions described above we introduce four alternative methods with the aim of improving efficiency:

- To speed up the insertion of elements in a BF the *Double Hashing technique* (DH) [9] has also been evaluated. Two initial digests h_1 and h_2 are computed. The final hash h_i is the result of an iterative linear combination of h_1 and h_2:

$$h_i = (h_1 + i \cdot h_2) mod\, n \tag{1}$$

  where n represents the hash output in $\mathbb{N}$.
- Depending on the size of the BF a different number of bits is required at the output of the hash function. To reduce the output we follow the *lazy mod mapping technique* [7]. It consists on applying a modular operation to the hash output taking the size of the filter as a parameter.
- Moreover, *truncated message digests* have been evaluated in order to use only one cryptographic hash to obtain all required hashes. This method specified in the NIST Standard FIPS 180-4 [16] proposes to take the necessary left most bits of a digest to reduce the size of the hash output. In our case, we take the necessary left most bits to compute as many positions as required for each element to insert to the filter.

– Finally, since multiple hashes are required per keyword we use the symmetric key to randomize the input of the hash function and prevent dictionary or pre-computed tables attacks. Message Authentication Code (MAC) constructions like the HMAC family of functions may also be considered for this purpose.

During this research we prioritized implementations adapted for processors with a low number of bit registers. In the case of cryptographic hash functions we did not find implementations designed for processors of 16 bits. MD5, SHA1 and SHA2 are adapted for 32 bit registers while SHA3 is adapted to 64 bit registers. Furthermore, SHA2 and SHA3 use 256 bit key lengths. It is remarkable to mention that there are a limited number of implementations of SHA3 due to its novelty.

3.3 Encryption Algorithms and Fast Hash Functions

As far as we know, no previous research based on applying SSE using BFs has proposed the use of an encryption algorithm plus a fast hash to fill in the filter. We propose to assess five widely used cryptographic algorithms with two different fast hash functions:

AES (1997). Today AES is the de facto standard [5]. It is an iterative block cipher based on a substitution-permutation network. It works with blocks of 128 bits and no major vulnerability has been unveiled so far. The number of rounds depends on the size of the key: 128/192/256 bit keys require 10/12/14 rounds respectively.

MISTY1 (1995). MISTY1 is a secure block cipher which uses a nested Feistel Network of a multiple of 4 rounds [12]. Recursively, each of these rounds uses a 3 round Feistel Network. It works with 64 bit blocks and sizes of 128 bit keys.

PRESENT (2007). PRESENT relies on a substitution-permutation network of 31 rounds for block sizes of 64 bits and key sizes of 80/128 bits [2]. PRESENT is considered secure although it has partially been broken for 26 rounds.

SKIPJACK (1998). SKIPJACK was designed to provide security on phones [6]. It is a block cipher that uses an unbalanced Feistel Network of 32 rounds and 80 bit keys. The most successful attack breaks 31 rounds but a full attack is not known to date. NIST recommends to avoid its use due to its weak key length.

XTEA (1997). XTEA is a block cipher that uses a 64 rounds Feistel Network and blocks of 64 bits with a 128 bit key [21]. Up to date, 36 rounds have been broken but is considered as secure in its full-fledge version.

FNV (1991). FNV is a non-cryptographic hash function based on an offset and a chosen prime that depends on the length of the output [8]. It is based on a Merkle-Damgård function that works byte by byte to obtain 32, 64, 128, 256, 512 or 1024 bit outputs.

Murmurhash3 (2010). This non-cryptographic hash function is based on a block inter-mixing [1]. Input bits are divided in blocks and simple operations on the block mix ensure that all blocks are affected by the precedent blocks. Bit outputs of 32 or 128 are allowed.

Additional Methods Used. For reducing the computation of encryption algorithms the *double hashing technique* has been used. In addition to this, to reduce the hash output length the *lazy mod mapping technique* has been applied. All these techniques are detailed in Sect. 3.2. Moreover, fast hash functions use different *offsets* and *seeds* to obtain different hash outputs.

In the case of AES we use a version designed for 8 bit registers in ECB mode with a 128 bit key. Besides, PRESENT uses a 80 bit key and is also adapted to 8 bits registers. SKYPJACK is adapted to 16 bit registers and the rest of algorithms to 32 bit registers. Moreover, fast hash functions used work with 32 bit lengths at the output.

4 Evaluation

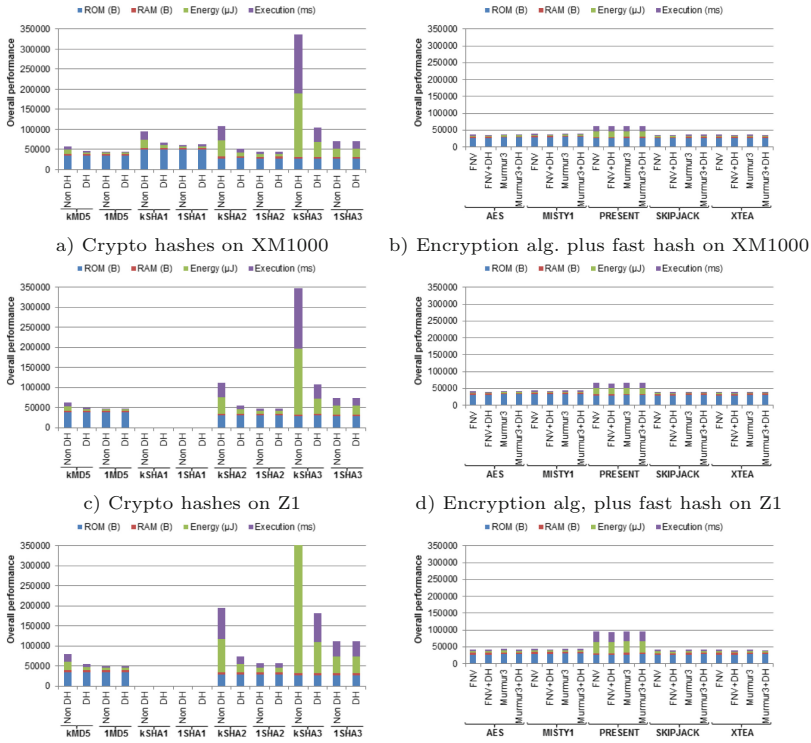
A real demo using wireless sensor devices for SSE has been implemented. Secure indexes and trapdoors based on BFs have been executed on motes which use the low power consumption IEEE802.15.4 standard at 2.4 GHz. The sensors assessed are Advanticsys XM1000, Zolertia Z1 and Crossbow TelosB. The evaluated algorithms have been programmed in C using the open source OS Contiki 2.6.

In our experiments 25 secure indexes of 128 bit positions are created. 10 elements of a few tens of characters are used to fill in each filter. From these elements 7 are used as real keywords for indexing the content of the document and the rest are chosen randomly to blind the filter. With these parameters we select an optimum number of 9 different hashes per keyword.

The parameters measured correspond to ROM (kB), RAM (kB), energy consumed by the CPU (μJ) and execution time (ms). As the following results show, the energy consumed by the CPU is proportional to the execution time.

4.1 Cryptographic Hash Functions

As detailed in Sect. 3.2 four cryptographic hash functions have been evaluated. If message digests are not truncated k hash functions are needed for each element, where k is the optimum number of hashes that in our case is 9. When truncating messages only the computation of one cryptographic hash function is needed for each element. Moreover, in all cases we considered the DH to evaluate the improvement achieved.



a) Crypto hashes on XM1000

b) Encryption alg. plus fast hash on XM1000

c) Crypto hashes on Z1

d) Encryption alg, plus fast hash on Z1

e) Crypto hashes on TelosB

f) Encryption alg. plus fast hash on TelosB

Fig. 2. Overall performance evaluation using (a, c, e) cryptographic hash functions and (b, d, f) encryption algorithms with a fast hash function.

Figure 2a shows the overall performance when using XM1000 devices. Figure 2c shows the results for Z1 motes and Fig. 2e for TelosB.

TelosB motes offer the poorest performance in all cases. XM1000 and Z1 sensors show similar performance with slightly better results in the first case.

When message digests are not truncated, the DH technique improves the performance of the system. If digests are truncated the performance is very similar but slightly better results are obtained whether the DH is not used.

In terms of overall performance we observe that truncated MD5, SHA2 and SHA1 are the best options for XM1000. Z1 and TelosB present a ROM overflow problem for SHA1, so only truncated MD5 and SHA2 functions are considered. SHA3 displays a poor performance when message digests are not truncated. In the case of TelosB we decided not to show the execution time for k SHA3 to improve the legibility of the figure. All these options are discussed in detail in Sect. 4.3.

4.2 Encryption Algorithms and Fast Hash Functions

In Sect. 3.3 we presented the five cryptographic algorithms used for the encryption of keywords and the two fast hashes required to insert them to a BF. In this section we have evaluated all these techniques plus the improvement obtained when using DH.

Figure 2b shows the overall performance when evaluating XM1000 devices. Figure 2d shows the results for Z1 motes and Fig. 2f for TelosB. At first sight, if we compare these results with the ones obtained for cryptographic hash functions we observe that the strategy of using encryption algorithms with a fast hash function improves the performance.

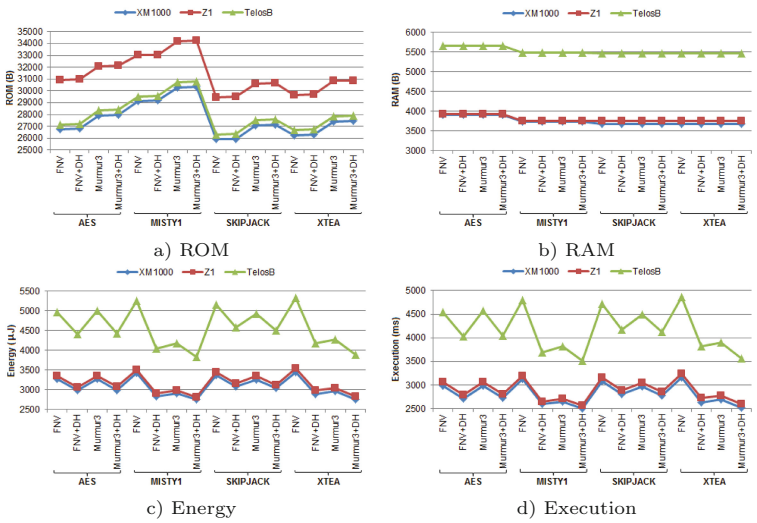
As in the previous section TelosB motes offer poor performance compared to the other two and XM1000 are slightly better than Z1.

In all cases the DH offers just marginally better results. Furthermore, similar overall results are obtained when using FNV and Murmurhash3. Concerning the cryptographic algorithms PRESENT offers the worst results for all devices. For this reason its evaluation is discarded in Fig. 3.

Figure 3a details the use of ROM. In all cases Murmurhash3 requires more ROM than FNV. Moreover, SKIPJACK and XTEA need less memory than the other options while MISTY1 requires more than the others. Besides, it can be observed that Z1 devices require more ROM than the other two.

The results for RAM are shown at Fig. 3b. Similar results are obtained in all cases except for AES that requires slightly more RAM. TelosB motes offer the poorest performance in this domain.

In terms of energy and execution (see Figs. 3c and d) all algorithms obtain better results when using Murmurhash3 except in the case of AES that improves



a) ROM b) RAM

c) Energy d) Execution

Fig. 3. Performance evaluation of the best cryptographic algorithms.

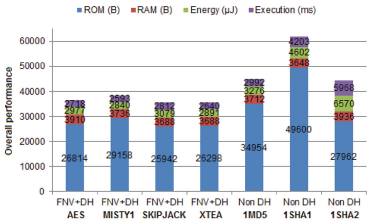
its execution when using FNV. MISTY1 and XTEA offer the best results when used with Murmurhash3 and the DH.

Taking into account the overall performance, we state that AES, MISTY1, SKIPJACK and XTEA with FNV and DH are the best options. As in the previous section, all these strategies are discussed in detail in Sect. 4.3.

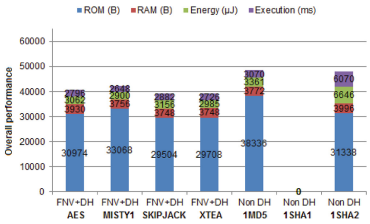
4.3 Best Strategies

In this section we discuss the practical use of the best cryptographic hashes and encryption algorithms plus a fast hash function. The results in terms of overall performance for XM1000, Z1 and TelosB devices (see Figs. 4a–c) show that encryption algorithms plus a fast hash function using the DH provide better results.

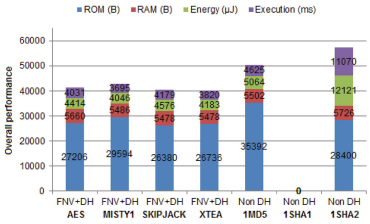
Figure 5a shows the ROM usage for the best strategies. It is remarkable to mention that SHA2 requires a similar memory amount than encryption algorithms. The results obtained for RAM (see Fig. 5b) indicate that all strategies require comparable resources. Finally, it can be stated that energy and execution



a) XM1000



b) Z1



c) TelosB

Fig. 4. Overall performance evaluation of the best strategies.

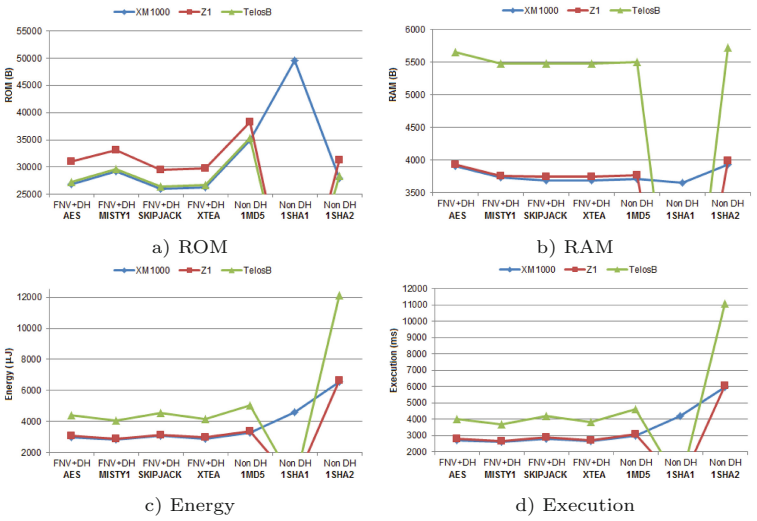


Fig. 5. Performance evaluation of the best strategies.

time (see Figs. 5c and d) required is higher for SHA2 and SHA1 while remains quite uniform in other cases.

In terms of security, we evaluate the suitability of the combination of encryption algorithms and a fast hash function. The selection of SKIPJACK is discarded due to its weak key length. MISTY1 offers poorer performance results when compared to AES and XTEA and with no security advantage, so we do not recommend its use either. The other encryption algorithms: AES and XTEA are considered secure. Nevertheless, AES is constantly subject to extensive analysis from the cryptographic community and consequently is considered as a highly resistant algorithm. For this reason, even if it displays a slightly worse behaviour than XTEA on sensors, we opt for the use of AES plus FNV and DH for constructing secure indexes and trapdoors.

If we compare the three cryptographic hash functions we observe that SHA1 offers the poorest performance. MD5 and SHA2 offer similar results but still they are paradoxically less performant than encryption algorithms. Concerning security, SHA2 is secure and MD5 is not collision resistant. This weakness does not affect security in our application due to the fact that collision attacks are typically used to impersonate someone but cannot guess the plaintext. Nevertheless, it must be taken into account that a theoretical pre-image attack has been discovered for MD5 and it is a matter of time before an attack that breaks this property in a reasonable amount of time is found. For this reason, we consider a truncated SHA2 without the DH the best option between all cryptographic hash functions.

Finally, we compare the best solution for each strategy defined: (1) AES plus FNV and DH and (2) a truncated SHA2 without the DH. We observe (see Fig. 4) that the first solution displays better results.

To summarize, we recommend the use of AES plus a fast FNV hash function using the DH due to its good performance and high level of security.

5 Conclusion

The aim of this research is to evaluate different strategies to allow the implementation of Searchable Symmetric Encryption techniques using wireless sensor devices. In this context, Bloom Filters are used to secure indexes and trapdoors related to encrypted documents saved on a cloud database.

The performance of mote devices is measured and compared in terms of ROM, RAM, CPU consumption and execution time.

Two different strategies are assessed. First of all, four well-known cryptographic hash functions are evaluated to save keywords in a filter. Furthermore, five widely used symmetric encryption algorithms combined with two different fast non-cryptographic hash functions are analyzed.

Our results show that the combination of an encryption algorithm with a fast hash function offers better results than using a cryptographic hash function. Based on our experiments, due to its higher performance on sensors and stronger level of security we recommend the use of AES plus FNV and the Double Hashing technique.

As future work we envision a strategy that allows the creation of secure indexes according to a certain entropy.

Acknowledgment. This work has been financially supported by the Swiss Hasler Foundation in the framework of the POPWiN project.

References

1. Appleby, A.: murmurhash3 (2011)
2. Bogdanov, A.: PRESENT: an ultra-lightweight block cipher. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 450–466. Springer, Heidelberg (2007). doi:10.1007/978-3-540-74735-2_31
3. Cazorla, M., Gourgeon, S., Marquet, K., Minier, M.: Survey and benchmark of lightweight block ciphers for MSP430 16-bit microcontroller. Secur. Commun. Netw. **8**(18), 3564–3579 (2015). http://dx.doi.org/10.1002/sec.1281
4. Eu-Jin, G.: Secure indexes. Technical report (2004). http://crypto.stanford.edu/eujin/papers/secureindex/
5. FIPS PUB 197, Advanced Encryption Standard (AES), National Institute of Standards and Technology, US Department of Commerce, November 2001. http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf
6. FIPS PUB 185, Escrowed Encryption Standard (EES). Federal Information Processing Standards Publication 185 (1994)
7. Fowler, G.: Fowler/Noll/Vo (FNV) hash (1991). http://isthe.com/chongo/tech/comp/fnv
8. Fowler, G., Noll, L.C., Eastlake, D.: The FNV non-cryptographic hash algorithm. Internet Draft (2015)

9. Kirsch, A., Mitzenmacher, M.: Less hashing, same performance: building a better bloom filter. In: Azar, Y., Erlebach, T. (eds.) ESA 2006. LNCS, vol. 4168, pp. 456–467. Springer, Heidelberg (2006)
10. Li, J., Chen, X., Xhafa, F., Barolli, L.: Secure deduplication storage systems with keyword search. In: 2014 IEEE 28th International Conference on Advanced Information Networking and Applications (AINA), pp. 971–977, May 2014
11. Li, T., Wu, Y., Zhu, H.: An efficient scheme for encrypted data aggregation on sensor networks. In: IEEE 63rd Vehicular Technology Conference, 2006. VTC 2006-Spring, vol. 2, pp. 831–835, May 2006
12. Ohta, H., Matsui, M.: A description of the MISTY1 encryption algorithm. RFC 2994, November 2000
13. Pub, N.: Draft FIPS pub 202: SHA-3 standard: permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication (2014)
14. Raykova, M., Vo, B., Bellovin, S.M., Malkin, T.: Secure anonymous database search. In: Proceedings of the 2009 ACM Workshop on Cloud Computing Security, pp. 115–126. ACM (2009)
15. Rivest, R.: The MD5 message-digest algorithm. Internet Request For Comments 1321 (1992)
16. Standard, N.S.H.: Federal information processing standards publication fipps 180–4 (2012)
17. Suga, T., Nishide, T., Sakurai, K.: Secure keyword search using bloom filter with specified character positions. In: Takagi, T., Wang, G., Qin, Z., Jiang, S., Yu, Y. (eds.) ProvSec 2012. LNCS, vol. 7496, pp. 235–252. Springer, Heidelberg (2012). doi:10.1007/978-3-642-33272-2_15
18. Tarkoma, S., Rothenberg, C., Lagerspetz, E.: Theory and practice of bloom filters for distributed systems. Commun. Surv. Tutor. IEEE **14**(1), 131–155 (2012). First
19. Wang, B., Yu, S., Lou, W., Hou, Y.T.: Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud. In: 2014 IEEE Conference on Computer Communications, INFOCOM 2014, Toronto, 27 April–2 May 2014, pp. 2112–2120 (2014). http://dx.doi.org/10.1109/INFOCOM.2014.6848153
20. Wang, C., Ren, K., Yu, S., Urs, K.: Achieving usable and privacy-assured similarity search over outsourced cloud data. In: Proceedings of IEEE INFOCOM, 2012, pp. 451–459, March 2012
21. Wheeler, D., Needham, R.: Tea extensions, also correction to XTEA, October 1998. www.ftp.cl.cam.ac.uk/ftp/users/djw3
22. Wu, Y., Ma, D., Li, T., Deng, R.: Classify encrypted data in wireless sensor networks. In: IEEE 60th Vehicular Technology Conference, VTC-Fall, vol. 5, pp. 3236–3239, September 2004