IET Computers & Digital Techniques

Special Issue: Defect and Fault Tolerance in VLSI and Nanotechnology Systems

IET Journals
The Institution of Engineering and Technology

RASSS: a hijack-resistant confidential information management scheme for distributed systems

ISSN 1751-8601
Received on 14th March 2018
Revised 12th November 2018
Accepted on 11th December 2018
E-First on 28th January 2019
doi: 10.1049/iet-cdt.2018.5167
www.ietdl.org

Lake Bu[1] ✉, Mihailo Isakov[1], Michel A. Kinsy[1]

[1]Adaptive and Secure Computing System Laboratory, Department of Electrical and Computer Engineering, Boston University, Boston, USA
✉ *E-mail: bulake@bu.edu*

Abstract: In distributed systems there is often a need to store and share sensitive information (e.g., encryption keys, digital signatures, login credentials etc.) among the devices. It is also generally the case that this piece of information cannot be entrusted to any individual device since the malfunction or compromising of one node could jeopardize the security of the entire system. Even if the information is split among the devices, there is still a risk when an attacker can compromise a group of them. Therefore we have designed and implemented a secure and robust secret sharing scheme to enable a more resilient sharing of confidential information. This solution provides three important features: (i) it uses threshold secret sharing to split the information into shares to be kept by all devices in the system; so the information can only be retrieved collaboratively by groups of devices; (ii) it guarantees the privacy of the confidential information under a certain number of passive hijacking attacks; and (iii) it ensures the integrity of the confidential information against any number of hijackers who actively and collusively attack the devices. It is able to identify all the compromised devices, while still keeping the secret unforgeable to attackers.

1 Introduction

Distributed systems have greatly impacted people's lives, from the internet of things (IoT) systems in industrial applications, communication and transportation infrastructures, and healthcare to cluster and edge computing. Their introduction has brought new computing challenges, especially in the domains of privacy and security. In those systems with a security demand, there is often a need for sharing and storing confidential information (secret) to the devices. It can be encryption keys, digital signatures, login credentials or important account numbers etc. This piece of information (secret) will later be used by the devices or nodes for certain security-oriented applications.

Usually, in a distributed system, the transmission channel between a server (the dealer of the secret) and nodes (shared secret holders) is well secured by various techniques, and the nodes are always verified for its identity by authentication. However, these protections cannot ensure the security of the secret when some of the nodes are hijacked by attackers. Through hijacking one or more nodes, an attacker will be able to silently acquire all the information it owns, and use it to conduct malicious actions with its legal identity. In addition, surveys [1] have shown that a huge number of commercialised devices in distributed systems have been using the same secret (cryptographic keys) on every device. Hence hijacking one device will lead to the ownership of thousands of other devices.

Obviously, this piece of secret cannot be entrusted as a whole to any individual device, because the malfunction of a single device will possibly jeopardise the security of the entire network. Thus the threshold secret sharing (TSS) scheme is often adopted as a fitting solution to this severe attack scenario. The confidential information (secret) of a network, i.e. secret keys, accounts, permissions, emergency response codes etc. is shared to all the devices in a manner that it can only be retrieved collaboratively by a group of devices. The minimum size of such a group is called 'threshold t', which draws the line of the system's privacy level. Below the threshold, the secret is information theoretically safe and kept private from retrieval.

Practical secret sharing techniques are deployed in many real world applications especially the distributed systems because of their matching nature. The most common example is the key management in wireless sensor networks. Rather than entrusting the cryptographic key to a single node, which can be easily compromised in hostile environments, the key is shared to a group of nodes and can only be retrieved collaboratively [2] to be used for digital signature or other cryptographic purposes at the other terminal. If some nodes are found to be malfunctional, then they will be revoked and replaced by the same number of healthy nodes to reach the threshold. The 'Vanish' project [3] uses the threshold property to make the secret key in a distributed system vanish when the number of shareholding nodes gradually decrease to below the threshold. Another application is in the hardware security module (HSM)-based systems. HSMs are widely used in bank card payment systems. Some HSMs [4] are produced and distributed by certification authorities and registration authorities to generate and share important secret keys under public key infrastructure. These HSMs also require implementation of a multi-part user authentication scheme, namely TSS. The most well-known application is probably the domain name system (DNS) security [5], which ensures the DNS servers to connect the users and their internet destinations (uniform resource locators and internet protocols) in a secure and verified manner. It has its root key split and shared among seven holders all over the world. In the case of an attack, if any five or more of the holders are able to come to a U.S. base, then they can reconstruct the root key using their shares to restore the internet connections. Technology survey companies also use TSS to store sensitive survey data to prevent them from being extracted by any single data analyst without the participation of others [6].

Although this technique reduces the risk of losing all the confidential information under the malfunction of one of a few devices, there is still the danger of attackers compromising a larger group of them. Due to this distributed nature, TSS schemes are susceptible to a number of attacks, such as passive attacks, man-in-the-middle (MITM) or share manipulations, i.e. cheating. The resulting share disclosure or distortions from these attacks may lead to the leakage of the original secret or retrieval of a wrong secret. Generally speaking, the TSS is able to maintain the privacy of the secret information under the existence of a small number (below the threshold) of cheaters. It alone does not guarantee the integrity of the secret. Although there have been a number of secure TSS schemes, they are often limited in their cheater tolerance. Generally, when the number of cheaters exceeds their

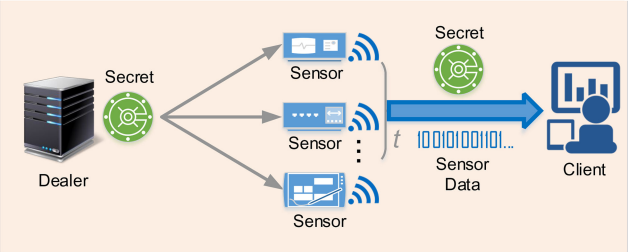

Fig. 1 *Three layers of the Odysseus system: the dealer who deploys the boards and the secret, the sensor boards as the shareholders with wireless communication capability, and the client(s) who collects the data as well as the secret*

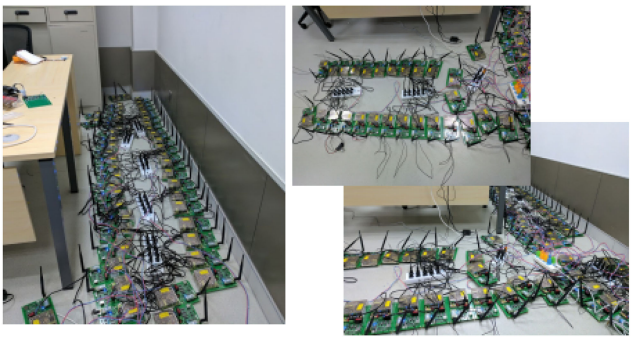

Fig. 2 *Prototype boards of Odysseus*

fault tolerance, neither the privacy or the integrity of the secret can be protected.

Therefore we propose a robust adaptive secure secret sharing (RASSS) [7] scheme to facilitate the sharing and storing of the confidential information in distributed systems. The major contributions of this scheme are as follows:

i. This scheme uses TSS to split the secret into shares to be kept by all devices in the system. Thus, the secret can only be retrieved collaboratively by groups of devices.
ii. Additional security features are added to the original TSS functionality to protect the privacy of a secret even when attackers have hijacked a group of devices to retrieve it.
iii. Besides privacy, this scheme ensures the integrity of a secret when there is a large amount of sophisticated and collusive attackers who have manipulated the shares to forge fake secrets.
iv. This scheme is able to detect and identify a large number of cheating or compromised holders up to the theoretical upper bound.
v. An automation tool is provided to conveniently generate the entire secret sharing and the accessory system based on user-specified parameters.

In this study, we will also introduce our own Odysseus IoT system, which requires a secure information sharing mechanism and has been the major motivation of designing the proposed scheme. It serves as a concrete example to illustrate the attack models and the defences. It is worth noting that the proposed scheme is a generalised technique and its application is not limited to the Odysseus system or distributed edge computing platforms.

The rest of the paper is organised as follows. Section 2 introduces the details of the Odysseus platform, as well as the original TSS scheme. We also define the attack model. Section 3 summarises related existing secure protocols for the TSS. Section 4 follows up with their vulnerabilities under the attack model. Section 5 introduces our new secure and robust secret sharing scheme, as well as a cheater identification protocol, followed by the deployment of the scheme on the Odysseus-distributed edge computing system. Section 6 presents a prototyped design automation tool to facilitate the systematic construction and analysis of the hardware-root-of-trust support for the algorithm. Finally, Section 7 concludes the paper.

2 Odysseus IoT system, the original TSS scheme, and the attack models

In this section, we will firstly introduce our distributed system named Odysseus, on which our proposed secure TSS scheme will be applied to. Without loss of generality, our scheme will be introduced and illustrated in the context of Odysseus.

2.1 System model – Odysseus IoT

The Odysseus computing architecture is a three-layer distributed system. It consists of (i) a set of distributed edge computing nodes connected to data transmission bases using a short-distance communication protocol (e.g. Zigbee), (ii) a pool of the data transmission bases to perform long-distance transmissions (e.g. WiFi), and (iii) a server-based backend compute infrastructure. The Odysseus edge computing nodes are sensor hosting boards with an open-interface and can support a multitude of programmable sensors. These sensors can either be heterogeneous or homogeneous. Users can select whichever sensors to be installed to the boards via general purpose input/output ports, based on the targeted application, before deploying the boards. An example of Odysseus' application is on fire-fighting and rescue: heat sensors to map the fire intensity and locations in a building, and motion sensors to identify human presence.

The original motivation of developing a secure and robust TSS is to protect our Odysseus IoT system. In the Odysseus, the dealer is the service provider who provides the Odysseus boards and is responsible for the deployment. The dealer (administrator) of the Odysseus system will deploy to a region a large number of sensor boards, and their sensor data can be requested remotely by different clients. From time to time, a client will request sensing data from a group of sensors, while retrieving from them a secret if necessary. The secret, such as an encryption key, a signature, or a login credential etc., will be used by the client on various applications associated with the sensor data. The system chart and prototype of Odysseus are shown below in Figs. 1 and 2.

The security issue of this platform also needs to be addressed. Although the dealer and clients alone can be trusted, the sensor hosting boards scattered all over a region are not physically monitored. Since any number of them can be subject to passive or active attacks, no critical information such as the secret should be entrusted to any individual board. There is even a danger of a large amount of them being hijacked by the attackers, meaning the adversary can gain full access to those devices. Therefore, there is a demand for a secure protocol to attain the privacy and integrity of the secret, as well as error tolerance under the existence of compromised boards.

2.2 Original TSS

Due to the distributed nature of Odysseus (and other distributed networks), there is often a demand to split the confidential information among the devices instead of entrusting the whole secret to every individual, such that the defect of a single device will not harm the security of the entire network.

The following notations are used to describe and evaluate the original TSS scheme, as well as the related secure variations:

- S: The original secret (a piece of confidential information).
- D_i: The public ID of the ith shareholder.
- h_i: The share of S to the ith shareholder.
- t: The threshold of a secret sharing scheme.
- c_{est}: The number of estimated cheaters.
- c_{act}: The number of actual cheaters.
- n: The total number of shareholders involved in a computation.
- b: The number of bits in a vector variable.
- $\oplus$: The addition operator in finite fields.
- $\cdot$: The multiplication operator in finite fields.

- $\oplus$: The cumulative sum operator in finite fields.
- $\prod$: The cumulative product operator in finite fields.
- MAC(): A secure message authenticating function.
- ENC(): A cryptographic encryption function.
- EtM(): An encrypt-then-MAC function.
- K: The cryptographic key.
- $\|$: The concatenation operator.
- E: The encoded secret where $E = \mathrm{EtM}(S, K)$.
- $\sim$: The vector distortion symbol.
- P_{miss}: The probability of failing to detect the conduct of cheating in a distributed system.

The concept of t-TSS was first introduced by Shamir [8] in 1979. All the computations should be carried out over Galois finite field (GF) arithmetic in order to maintain the information theoretic security. To share a secret S, a polynomial of degree $(t-1)$ is used to compute and distribute the shares, where the secret S serves as the free or leading coefficient, and all other coefficients can be arbitrarily chosen. The shares are the evaluations of the polynomial by each holder's D_i.

The share distribution equation when S is placed as the free coefficient is as follows:

$$h_i = S \oplus a_1 D_i \oplus a_2 D_i^2 \oplus \cdots \oplus a_{t-1} D_i^{t-1}.$$

Also, as the leading coefficient

$$h_i = a_0 \oplus a_1 D_i \oplus a_2 D_i^2 \oplus \cdots \oplus S D_i^{t-1}, \tag{1}$$

where $S, h_i, D_i \in \mathrm{GF}(2^b)$.

The ID number D is publicly known to everyone while the share h is kept private by each shareholder.

With any subset of at least t shareholders' IDs and shares, one can use the Lagrange interpolation formula to reconstruct the secret.

If S is placed at the free coefficient, it can be retrieved by

$$S = \bigoplus_{i=0}^{t-1} \frac{D_i \cdot h_i}{\prod_{j=0, j\neq i}^{t-1} (D_i \oplus D_j)}.$$

If S is the leading coefficient, it can be retrieved by

$$S = \bigoplus_{i=0}^{t-1} \frac{h_i}{\prod_{j=0, j\neq i}^{t-1} (D_i \oplus D_j)}. \tag{2}$$

Such a construction is $(t-1)$-private. This means it needs at least t shareholders to reconstruct the secret and so any $(t-1)$ or fewer shareholders have zero knowledge of the secret.

For computation simplicity, in this study, we choose to place S as the leading coefficient as shown in (2). We also assume that the system works over a finite field $\mathrm{GF}(2^b)$, where in most computer systems, $b = 32, 64, 128, 256, \ldots$.

The original scheme's share distribution and secret reconstruction procedures are shown in Fig. 3, which matches with the Odysseus and many other distributed architectures very well in the administrator–devices–clients three-layer structure.

Remark 1: Shamir's secret sharing scheme is supposed to work under finite field arithmetic where the field size should be a prime or power of a prime. Ordinary arithmetic will be vulnerable and any secret can be retrieved by at most two carefully selected shareholders instead of t.

In the ordinary positive integer arithmetic, for instance, if a shareholder's ID is $D_i = 1$, their share will be $h_i = a_0 + a_1 + \cdots + S$, namely the sum of all coefficients of (2). Also, in the ordinary arithmetic, it is obvious that $h_i > a_l | a_l \in \{a_0, a_1, \ldots, S\}$. Then this holder can find another holder with ID $D_j \geq h_i$ whose share is hj. If these two shareholders collude they will easily get the secret

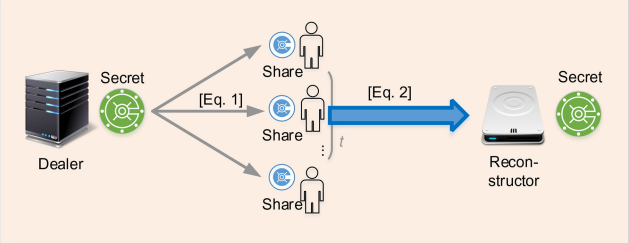

Fig. 3 *Secret sharing and reconstruction flow. The reconstructor can be omitted if there is no end user and every shareholder (either device or person) has trustworthy computation capability*

regardless of the t by expressing h_j in the radix of D_j, where the most significant digit will be S.

However, in a finite field or modular arithmetic, one can never have $h_i > a_l | a_l \in \{a_0, a_1, \ldots, S\}$ if $h_i = a_0 \oplus a_1 \oplus \cdots \oplus S$.

2.3 Attack model

We define the attack model below which is much stronger than what the original scheme and its conventional secure variations can deal with.

Definition 1: The attack model in this study is described by the following characteristics:

i. The dealer and the clients are trusted.
ii. The shareholders (devices in a distributed system) are not trusted and there is no limit to the number of compromised devices or cheaters.
iii. The cheaters are able to gain full control of the hijacked devices, meaning to read its memories, input/output ports or to tamper them.
iv. The cheaters can also eavesdrop or tamper the communication channels between devices, and dealer and clients.
v. The attackers have the knowledge of the system's basic parameters $(n, t,$ equations [1, 2] etc.). They can work collaboratively.
vi. The goals of the attackers are:
    a. *Passive attack:* to stealthily compute and acquire the original secret.
    b. *Active attack:* to select their own secret and submit it to the clients without being spotted.

Note: Besides the shares, each Odysseus board also submits its sensor data to the clients. However, those are the source data and their verification is another issue beyond the scope of this study.

When the cheaters work collusively, they are able to share any information they hold or to modify it according to their common interest. We also assume that the cheaters have sufficient computation power to calculate equations such as [1, 2] and other necessary tasks.

3 Conventional secure protocols for TSS

In this section, we will introduce the existing secure protocols against passive and active attacks to TSS. In the following section, their vulnerabilities under the attack model defined previously will be explored.

3.1 Secret privacy preservation

The property of TSS only allows t or more shareholders (devices) to retrieve the secret. Below this threshold, the secret information is theoretically secure. Namely, $t-1$ devices have no more knowledge of the secret than any individual device does. However, if the cheaters have compromised t or more devices, which is $c_{\mathrm{act}} \geq t$, then the privacy of the secret is not guaranteed since they can use (2) to retrieve it.

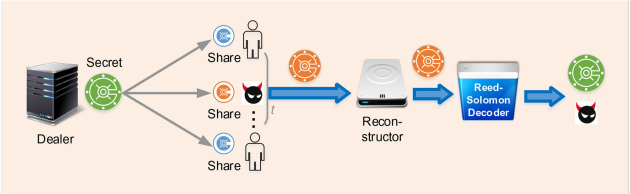

Fig. 4 *RS decoder is not only able to correct the distorted shares, but also identify the malicious shareholders with their IDs, as long as (3) holds*

3.2 Approaches to resist share tampering

After the invention of Shamir's secret sharing, it was noticed that if any number of shareholders participating in the secret reconstruction apply an active attack by changing their shares to make h_i to $\tilde{h}_i \neq h_i$, the retrieved secret will be distorted $\tilde{S} \neq S$ according to (2). Therefore, the authenticity of the submitted shares or the retrieved secret needs to be verified.

3.2.1 Share verification: Researchers [9–11] have proposed approaches to verify the validity of shares with a probability of 1. The common feature in the latter approaches is that, if the shares can be encoded to a codeword of a certain error control code (ECC), then the codeword's symbols (shares) can be verified and corrected within the ECC's capability.

Particularly, the share distribution (1) is inherently equivalent to the non-systematic encoding equation of the well-known Reed–Solomon (RS) ECC codes. RS codes are maximum distance separable (MDS) codes which meet the Singleton bound with equality. With such a distribution equation, an (n, t, d) RS codeword $(h_0, h_1, \ldots, h_{n-1})$ is encoded with n symbols (shares) in total, t information symbols, and distance $d = n - t + 1$, which corrects up to $((d-1)/2)$ (or $((n-t)/2)$) erroneous symbols with algorithms in [12, 13].

In the secret sharing language, with n shareholders' IDs and shares, we are able to tolerate up to $c_{\text{est}} \leq ((n-t)/2)$ shares maliciously modified by cheaters. Theoretically speaking, the error correction capability of RS codes can tolerate up to $c_{\text{est}} < n/2$ cheaters if $n \gg t$. However, oftentimes an assumption is made that there should be $c_{\text{est}} < t$ cheaters such that a group of all cheaters has no access to the secret [14]. Then we have

$$c_{\text{est}} < n/3. \tag{3}$$

If n instead of t shareholders are involved in the share error correction by RS decoders, then the correctness of the retrieved secret is ensured when (3) holds as shown in Fig. 4. Consequently, the secure secret sharing is both $(t-1)$-private and $(t-1)$-resilient that up to $t-1$ shareholders cannot reconstruct the secret, and up to $t-1$ cheaters cannot affect the correctness of the secret [15].

3.2.2 Secret verification: Besides share verification with share correction probability of 1, another approach is to sign the original secret with a key K using a message authentication code (MAC) function. Then the original secret is shared together with its MAC (usually in a manner of concatenation) to the holders. Denoting the encoded secret as $(S \parallel \text{MAC}(K, S))$, then (1) becomes

$$h_i = a_0 \oplus a_1 D_i \oplus a_2 D_i^2 \oplus \cdots \oplus (S \parallel \text{MAC}(K, S)) D_i^{t-1}. \tag{4}$$

At the reconstructor end, after the retrieval of the possibly distorted $(\tilde{S} \parallel \widetilde{\text{MAC}(K, S)})$, the following authentication equation is evaluated:

$$\text{MAC}(\tilde{K}, \tilde{S}) \stackrel{?}{=} \widetilde{\text{MAC}(K, S)}. \tag{5}$$

An inequality indicates the detection of cheating. If this MAC function has a high enough security level, such as 2^{-128} (or lower) collision or mis-detection probability, then it is generally believed that all distortions will be spotted. The secure protocol of Shamir's secret sharing is shown below.

There are commonly two ways to sign the original secret: hash-based MAC (HMAC) with a key, and algebraic manipulation detection (AMD) codes with a random vector.

A. HMAC with a key: HMAC, keyed-hashing for a message authentication code, is the most often used technique for authentication nowadays. To sign a secret S, the nested equation is defined as follows [16]:

Definition 2: Let HMAC() be the HMAC function, K the signing key, and K' be derived from K by padding to the right zeros to the block size. Also, let H be a hashing function; opad, the outer padding; and ipad, the inner padding. Then

$$\text{HMAC}(K, S) = H((K' \oplus \text{opad}) \parallel H((K' \oplus \text{ipad}) \parallel S)). \tag{6}$$

The client can authenticate the secret using the HMAC version of (5):

$$\text{HMAC}(\tilde{K}, \tilde{S}) \stackrel{?}{=} \widetilde{\text{HMAC}(K, S)}. \tag{7}$$

With secure hash algorithm-2 256 or higher used for $H()$ [17], the collision rate is $< 2^{-128}$ and considered cryptographically secure.

B. AMD with a random number: Cramer *et al.* [18] have proposed an AMD code to detect any modification of secrets with a probability close to 1. Karpovsky *et al.* [19] later generalised this code with flexible construction.

Unlike HMAC, it operates over finite fields and its security level is adjustable by block size b. The AMD encoding is defined as follows:

Definition 3: Let $K = (K_1, K_2, \ldots, K_m)$, where $K_i \in \text{GF}(2^b)$ is a randomly generated b-bit vector. An gth order generalised Reed–Muller code with m variables consists of all codewords $(f(0), f(1), \ldots, f(2^{bm}-1))$, where $f(K)$ is a polynomial of $K = (K_1, K_2, \ldots, K_m)$ of degree up to g. Let

$$A(K) = \begin{cases} \oplus_{i=1}^{m} K_i^{g+2}, & \text{if } g \text{ is odd;} \\ \oplus_{i=2}^{m-1} K_1 K_i^{G+1}, & \text{if } g \text{ is even and } m > 1; \end{cases}$$

where $\oplus$ is the accumulated sum in $\text{GF}(2^b)$. Let

$$B(K, S) = \bigoplus_{1 \leq j_1 + j_2 + \cdots + j_1 \leq g+1} y_{j_1, j_2, \ldots, j_m} \prod_{i=1}^{m} K_i^{j_i},$$

where $\prod_{i=1}^{m} K_i^{j_i}$ is a monomial of R of a degree between 1 and $g+1$. Also, $\prod_{i=1}^{m} K_i^{j_i} \notin \Delta B(K, S)$, which is defined by

$$\begin{cases} \{K_1^{h+1}, K_2^{g+1}, \ldots, K_m^{g+1}\}, & \text{if } g \text{ is odd;} \\ \{K_2^{g+1}, K_1 K_2^{g}, \ldots, K_1 K_m^{g}\}, & \text{if } g \text{ is even and } m > 1. \end{cases}$$

Let $f(K, S) = A(K) \oplus B(K, S)$, then a generalised AMD codeword is composed of the vectors $(S, \boldsymbol{K}, f(K, S))$, where S is the information portion, $\boldsymbol{K}$ the random vector, and $f(K, S)$ the redundancy signature portion [19].

Remark 2: If the attack involves a non-zero error on the information S, which is the major purpose of almost all attacks, then in $f(K, S)$ the term $A(K)$ can be omitted [20]. Furthermore, if only one random number vector is used, the encoding equation can be further more simplified to

$$\begin{aligned} \text{AMD}(K, S) &= f(K, S) \\ &= \bigoplus_{1 \leq j_1 + \cdots + j_i + \cdots + j_m \leq h+1} S_{j_1, \ldots, j_i, \ldots, j_m} K^{j_i}, \end{aligned} \tag{8}$$

where S_{j_i} is a b-bit block of S.

The client can authenticate the secret using the AMD version of (5):

$$\mathrm{AMD}(\tilde{K}, \tilde{S}) \stackrel{?}{=} \overline{\mathrm{AMD}(K, S)}. \quad (9)$$

The probability of mis-detecting a distortion of S in (9) is upper bounded by $g/2^b$ [18], where g usually is a very small number in most constructions. With b selected to be 128 bits or larger, the security level of AMD codes will be in the same order of HMAC (2^{-128} or less in attack mis-detection rate).

Note: Although HMAC and AMD codes are different approaches for authenticating the retrieved secrets, there is no essential difference in their design philosophy as (7) and (9) have shown.

It is notable that there are two potential drawbacks on the secret verification approach. Firstly, in the previous works, there was no explanation on how to securely transmit the MAC key K from the dealer to the client. Secondly, with this approach alone it can only detect the distortion of the secret, but not identify the cheaters nor retrieve the correct secret.

4 Vulnerabilities of the conventional secure TSS schemes

In this section, we will illustrate the vulnerabilities of the conventional secure schemes under the attack model defined previously. Owing to the scattering nature of many distributed systems, it is not unusual to have an unexpected scale of attacks beyond the estimation. The demand for a more secure and robust confidential information sharing scheme for distributed systems is the major motivation of designing the proposed solution in the next section.

4.1 Attack model

We firstly define the attack model, which is an indicator of the attackers' capability. The most commonly seen attacks on secret sharing schemes are share distortions, which could be caused by dishonest shareholders (cheaters) who maliciously change their shares, or by a MITM who tampers the shares of honest shareholders. Usually, people do not distinguish the two since both of them will result in share manipulations which lead to the retrieval of a wrong secret.

Definition 4: The attack model is described by the following characters:

i. The dealer and the client are trusted.
ii. The devices are not trusted. They could either be dishonest or attacked by MITM. We will refer to both as cheaters. For the simplicity of later expressions, we assume for each distorted share, there is a cheater to associate with.
iii. The cheaters have the knowledge of the system's basic parameters ($t, b, D_i, c_{\mathrm{est}}$ etc.) and they can work collaboratively to manipulate the shares to forge a new secret.
iv. The channel for the cryptographic key between the dealer and the client could be subject to MITM.
v. The goal of the attackers is to learn the original secret or forge their own secret without being spotted.

Based on the capability, there is a wide range of attacks applicable to a TSS system, even if it is protected by conventional security approaches.

4.2 Passive attack: acquiring the original secret

Usually, an assumption has to be made that $c_{\mathrm{est}} < t$ so that a group of all cheaters cannot retrieve the secret by themselves. However, it could happen that there exist more cheaters than originally estimated, such that $c_{\mathrm{act}} \geq t > c_{\mathrm{est}}$. With any t of them, it is easy to acquire the original secret by (2).

4.3 Active attack: making the secret unaccessible

Here we assume the distributed system's TSS is already equipped with the share verification module. As mentioned in Section 3.2.1, the essence of such a module is to encode the shares into a codeword, whose validity can be verified by the RS decoding algorithm. Although RS codes are known for their strong error correction (tolerating $c_{\mathrm{est}} < n/3$ cheaters), their encoding procedure is linear and susceptible to cheating exploits.

If the number of cheaters satisfies ($n/3 < c_{\mathrm{act}} < n - t + 1$), although the RS decoder can still raise an alarm for cheating, it will be beyond its share error correction capability. Therefore the system is unable to retrieve the secret or identify the cheaters.

4.4 Active attack: forging a legal secret

If the number of cheaters satisfies ($n - t + 1 \leq c_{\mathrm{act}} \leq n$), they will be able to manipulate the entire system. For instance, the cheaters can pick another share distribution polynomial different from (1) with random coefficients b_i and their own forged secret $\tilde{S}$:

$$h_i' = b_0 \oplus b_1 D_i \oplus b_2 D_i^2 \oplus \cdots \oplus \tilde{S} D_i^{t-1}. \quad (10)$$

The new shares h_i' of the cheaters will be the evaluation of (10) by the same IDs D_i. When $c_{\mathrm{act}} \geq n - t + 1$, the cheaters' shares will form a new legal RS codeword, which will never be detected by the RS decoder. The secret reconstruction will then submit to the client the secret $\tilde{S}$ that the cheaters have selected. If the client uses the forged secret on their important applications, such as digital signatures, the attackers can effortlessly break those applications.

Example 1: A secret sharing system has a secret $S = 111$ in the GF(2^3) finite field. It requires $t = 2$ shareholders to reconstruct the secret every time. The following share distribution polynomial is used to generate the shares:

$$h_i = a_0 \oplus S D_i = 010 \oplus 111 D_i.$$

The protocol is designed in such a way that up to one cheater can be tolerated. Therefore, in the secret reconstruction stage, there will be $n = 3c_{\mathrm{est}} + 1 = 4$ shareholders involved. Suppose that in the secret reconstruction, shareholders with IDs $D_0 = 001, D_1 = 010, D_2 = 011, D_3 = 100$ are involved. Also, the shares distributed to them are $h_0 = 101, h_1 = 111, h_2 = 010, h3 = 001$. These four shares form a legal RS codeword $v = (101, 111, 010, 001)$ with distance $d = n - t + 1 = 3$ and it can correct up to one error.

In the case where all four of them are cheating collusively and they have selected their own secret key $\tilde{S} = 100$, the share distribution polynomial will be

$$h_i' = b_0 \oplus \tilde{S} D_i = 001 \oplus 100 D_i.$$

Thus their shares will be maliciously changed to $h_0 = 101, h_1 = 010, h_2 = 110, h3 = 111$, which is also a legal codeword $v' = (101, 010, 110, 111)$ of an $(n, t, d) = (4, 2, 3)$ RS code. This codeword will be considered as a valid codeword by the RS decoding algorithm [13] and there will be no cheating alarm. As a result, the fake secret $\tilde{S} = 100$ is retrieved by those shares under (2). During the entire procedure, the cheating will not be detected.

4.5 Active attack: framing up the honest shareholders

Another vulnerability that cheaters can exploit when ($n - t + 1 \leq c_{\mathrm{act}} \leq n$) is to frame up the honest shareholders so that the decoder treats the honest parties as 'cheaters' and cheaters as 'honest shareholders'. If c_{act} is large enough that the number of honest shareholders is $n - c_{\mathrm{act}} \leq ((n - t)/2)$, then the honest shareholders are within the RS decoder's error correction capability. Since all cheaters' shares are generated by the same

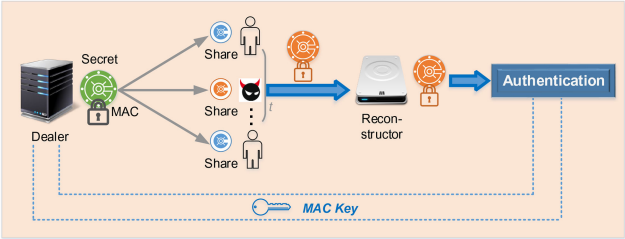

Fig. 5 *Secret sharing scheme with secret authentication in the context of Odysseus system*

forged secret sharing polynomial, the honest minority will be treated as cheaters and 'corrected'. The cheaters' fake secret will be regarded as the valid secret as the result of (2).

Example 2: Suppose that we have the same secret sharing system as in Example 1. Let us have three shareholders $\{D_0 = 001, D_1 = 010, D_2 = 011\}$ as cheaters, and shareholder $D_3 = 100$ is an honest participant. The codeword for the shares submitted to the RS decoder will be $v' = (101, 010, 110, 001)$. v' will be decoded as $(101, 010, 110, 111)$ which is the cheaters' codeword. Shareholder $D_3 = 100$ will be labelled as a 'cheater'. Consequently, the forged secret $\tilde{S} = 100$ (as in Example 1) will be retrieved.

4.6 Active attack: against secret verification

For the distributed system with TSS equipped with secret verification, as mentioned in the previous section, although it has a high probability of detecting any number of share distortions, it alone is not able to identify the cheaters nor correct the shares. In addition, there is one more problem that has to be addressed: how to securely pass the MAC key K from the dealer to the client (as in Fig. 5) in order to conduct the secret authentication, giving that the transmission channel might have eavesdropped.

There can be more types of attacks besides the ones listed above. Especially when the number of cheaters is beyond estimation, the entire system can be subject to total manipulation. Therefore there is a demand for a more secure and resilient scheme to handle the severe attacks.

5 Secure and robust secret sharing scheme for distributed systems

In this section, we propose a new secure and robust secret sharing scheme for distributed systems. Comparing with the current secret sharing scheme, which has limited protection against the cheaters, the advantages of the proposed scheme are as follows:

i. The proposed scheme protects both the confidentiality and the integrity of the secret.
ii. The proposed scheme is able to detect and identify the cheaters up to the theoretical upper bound.
iii. The proposed scheme uses the physical unclonable functions (PUFs) to ensure the security of the cryptographic key update.
iv. The proposed scheme works in an adaptive manner that a more powerful module will only be activated when the previous module fails. Thus the scheme functions in a cost-efficient way and consumes minimum resource on average.

The following subsections are organised in the order of an overview of the proposed scheme, a detailed introduction of the modules of this scheme, and finally a simple numeric example to demonstrate the scheme.

5.1 Overview of the proposed secure secret sharing scheme

The proposed scheme consists of five adaptive stages and one optional stage, in order to ensure the security of a TSS scheme under cheaters, while minimising the overhead.

Stage 1: dealer – encoding and distribution of the secret: First, the dealer will encode the secret S with an encryption-then-MAC function EtM() to $E = \text{EtM}(K, S)$, where K is randomly picked from the dealer's repository, which stores the challenge and response pairs (challenge response pair (CRP)s) of the client's PUF. Then the dealer distributes E using (2) to n shareholders. The detailed key transmission protocol will be introduced in the following subsections.

Stage 2: client – secret retrieving: The client will select an arbitrary set of t shareholders to participate in the secret retrieving using (2). The retrieved secret will be authenticated by (7) or (9) by the K generated at the client end. If the authentication claims the validity of the secret, then it is considered a successful secret reconstruction with no cheat. If not, the scheme calls for Stage 3 for share correction.

Stage 3: client – share error correction: This stage uses the RS error correction module in the classic protocol. Here, $n = 3c_{\text{est}} + 1$ shareholders will be invited to participate in the protocol, where c_{est} is the number of estimated cheaters defined by the system. The RS decoder will try to correct the shares and then send them back to the secret reconstruction and verification modules at the client end. If it passes both the share correction (by RS decoder module) and secret verification (by authentication module), then the secret reconstruction is successful. If either module fails then the protocol ascends to its fourth stage, indicating that the actual number of cheaters is greater than $n/3$.

Stage 4: client – small-scale group testing: In this stage, a small-scale group testing module is called to identify the cheaters mis-detected in Stage 3. This module generates a lightweight group testing pattern based on superimposed codes in order to identify the dishonest parties.

Stage 5: client – large-scale group testing: This stage with a large-scale group testing pattern will be activated if c_{act} is a large number and stage 4 fails. It will involve n shareholders, among whom there can be up to $c_{\text{est}} = n - t$ cheaters with a minimum number of t honest holders. This is also the theoretical upper bound of the number of cheaters when the secret is still retrievable. Even if there are more than $n - t$ cheaters, it is still able to detect cheating despite not being able to reconstruct the correct secret due to the lack of honest holders.

Stage 6: optional extra invitation: This optional stage is designed for clients who have access to larger resources than just n devices. By involving more devices, the chances of having more honest devices may increase. Thus, the client may be able to retrieve the correct secret, while identifying all the cheaters even if $c_{\text{act}} > n - t$.

The work flow of the proposed scheme is shown in Fig. 6. The techniques adopted in the six stages will be introduced in the following subsections.

5.2 Encoding of the secret

In order to perform obfuscation and authentication of the secret, we will apply the encryption-then-MAC function to encode the original secret S to E by

$$E = \text{EtM}(K, S) = \text{ENC}(K, S) \parallel \text{MAC}(K, \text{ENC}(K, S)). \quad (11)$$

The encryption function ENC() can be the standard advanced encryption standard (AES) or other lightweight approaches. Also, the MAC() function can be either HMAC with a fixed security level P_{miss} or AMD codes with flexible P_{miss} as mentioned in Section 3.2.1. AMD codes are able to trade-off between the security level and hardware cost conveniently by adjusting the vector size b, which sometimes is an ideal choice for distributed systems with limited resources.

For some distributed systems without a client end, it is not possible to maintain the confidentiality of the secret if there are more than t devices compromised. This is because the TSS scheme for this case entrusts the secret to the devices themselves. However, for other distributed systems, such as the Odysseus with a client end, it is possible to protect the privacy of the secret even if $c_{\text{act}} \geq t$

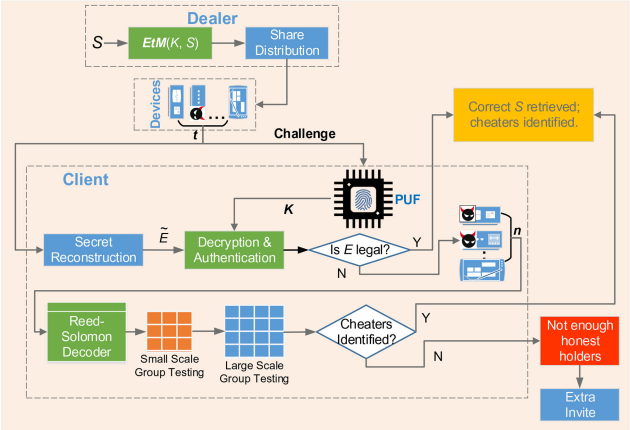

Fig. 6 *Proposed scheme functions with three entities: a dealer, a group of devices(shareholders), and a client. The dealer firstly encodes the secret with encryption then MAC (EtM) to preserve its confidentiality and authenticity. The secret and the EtM key's zero-knowledge clue, known as the challenge of a PUF, are both shared through the secret distribution equation to the devices. Then the client gathers t devices to retrieve the secret, as well as the EtM key from his PUF. If the secret authentication fails, then the RS decoder, small-scale group testing, large-scale group testing will be activated by order. The cost increases as a more powerful module is called. Yet the assurance of retrieving the correct secret and identify all the cheaters also increases*

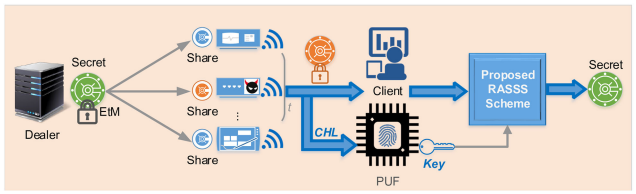

Fig. 7 *Dealer now shares both the encoded secret and the challenge to the devices. Once the client retrieves the secret, stages 2–4 in Section 5.1 will be performed to identify the cheaters (if any)*

because the secret is entrusted to the client. In this way, even if the attackers have compromised more than t devices, they can only acquire the cipher but not the secret plaintext. However, there is a critical issue of transmitting the EtM key to the client securely, which will be introduced in the following subsection.

5.3 EtM key transmission

The core of this proposed scheme's security is to establish a secure transmission channel for K which is

- *Eavesdrop resistant:* if the cheaters eavesdrop the channel, they should not acquire any knowledge of K.
- *Easy to update:* it should be easy and secure to update K on both the dealer and the client sides.
- *Unforgeable:* a cheater should not be able to predict, duplicate, or forge the keys.
- *Uniqueness:* in case of a multi-client secret sharing system, the different client should have different sets of keys.

Based on the criteria above, the PUF is an excellent and fitting solution. Another choice is to use public and private key pairs. Since Odysseus and many other distributed systems are hardware based, it is very convenient and natural to implement PUFs on them. Therefore, we will use PUFs to facilitate the transmission of K in this study. Although this concept of PUF has been known since 1983 [21], the term PUF only came to be in 2002 [22]. A PUF is a piece of hardware that produces unpredictable responses upon challenges due to their manufacturing variations. PUF has the property of *easy to make and hard to duplicate*, even under exact the same circuit layout and manufacturing procedures. A PUF can be made from a device's [either application specific integrated circuit or field programmable gate array (FPGA)] memory cells or circuits without modifying the device's architecture. Owing to its attributions of randomness and uniqueness, PUF provides an inexpensive and integrated solution for random number or secret key generation, dynamic authentication, and identification [23].

The PUF serves as a cryptographic primitive in a manner of challenge–response pairs (CRPs). Each PUF's output (response) is a non-linear function of the outside input (challenge) and the PUF's own physical, intrinsic, and unique diversity, in another word, 'silicon fingerprints' [24]. Given the same challenge, the same PUF design on different circuits will return different responses, which cannot be predicted by just having the challenge vector. Therefore PUF is an ideal choice in facilitating the transmission of K.

Algorithm 1: For the kth round of secret sharing, denote the secret as S_k, the arbitrarily selected challenge and response of the client's PUF as CHL_k and K_k, respectively. Then the EtM key K is transmitted from the dealer to the client as follows:

i. When a client registers to the dealer, the dealer challenges the client's intrinsic PUF with a set of inputs and stores its CRPs.
ii. Before S_k is to be distributed, the dealer selects an arbitrary CRP and uses its response K_k to encode the secret with EtM() to E_k. At the same time, the challenge CHL_k takes the position of the share distribution polynomial's free coefficient. Therefore (1) becomes

$$h_i = \mathrm{CHL}_k \oplus a_1 D_i \oplus a_2 D_i^2 \oplus \cdots \oplus E_k D_i^{t-1}. \quad (12)$$

Then the encoded secret E_k is distributed in the form of shares to the devices of the distributed system.
iii. When S_k needs to be retrieved, t holders will turn in their IDs and shares to the client;
iv. The client uses (2) to retrieve the encoded secret E_k, and by another Lagrange interpolation formula the client calculates CHL_k

$$\mathrm{CHL}_k = \bigoplus_{i=0}^{t-1} \frac{D_i \cdot h_i}{\prod_{j=0, j\neq i}^{t-1} (D_i \oplus D_j)}. \quad (13)$$

The client takes CHL_k to its PUF and regenerates the corresponding response K_k, which is the same key used by a dealer to EtM S_k. This K_k is used to authenticate and decrypt the retrieved encoded secret E_k.

Now the Odysseus system (or other similar distributed systems) equipped with the proposed scheme will have the following work flow (Fig. 7).

The advantage of this protocol is that CHL_k leaks no information of K_k. Even if there are t or more cheaters calculate CHL_k, they are still not able to acquire the corresponding K_k because the CRPs of a PUF is not predictable.

When a new secret S_l is about to be distributed, the dealer can select another CRP of the PUF to EtM the secret and embed the new challenge CHL_l to (12). This makes the update of the key to K_l simple and secure.

5.3.2 Selection of PUFs for secret sharing: Based on where the variation comes from, there are multiple types of PUFs. Delay PUFs and memory PUFs are the two popular implementations. Delay PUF uses the random variation in delays of wires and gates, and their race condition to generate the response bits. Memory PUF is based on the random initial state (1 or 0) of each memory cell. Depending on the size of the CRPs, there are weak and strong PUFs, which have different applications in security. Weak PUFs' CRP size grows linearly with the PUF size, while strong PUFs' CRPs exponentially.

Table 1 EtM Key exchange hardware costs

	PUF	PRNG	Key-exchange
LUT	1380	4841	15,232
register	1539	2844	13,311
slice	768	1521	5298
major Op	oscillation	multiplication	field exponentiation

[a]All three schemes are designed for the $E = 128$-bit EtM key update.

[b]A RO PUF is adopted. As for the PRNG, a Bernoulli chaotic map-based random number generator is implemented. The key-exchange is a standard two-party Diffie–Hellman scheme.

[c]The complexity of the major operations (Major Op) is different for the three algorithms. The key-exchange requires complicated field exponentiation operations, the PRNG standard floating point multiplications, and the PUF only clock-less oscillation of inverter rings.

In our design, we consider the frequent updates of the key (up to one key per secret). Thus we have selected the delay PUFs because of their large sets of CRPs. In our design, we use FPGAs to implement the secret sharing system with both the ring oscillator (RO) PUF based on the race condition of two ROs, and the Arbiter PUF based the delay difference between two MUX chains [25]. We also improved the design of both to increase the Hamming distance among the responses, while developing a design automation tool (introduced in Section 6).

5.3.3 Alternative methods for EtM key transmission: There can be alternative approaches to facilitate key transmission if it does not have to be hardware-based. There is one critical rule that these approaches always have to follow: the free coefficient in (12) needs to reveal zero-knowledge about K_k so that the passive attacks in Section 4.2 cannot apply. In this subsection, two possible alternatives are introduced.

Algorithm 2: This algorithm uses the classic cryptographic key exchange scheme. In this way, the framework of Algorithm 1 can be kept with a slight modification.

i. The dealer and the client firstly achieve an agreement on a secret a in a finite group GF(). This can be implemented by a secure eavesdrop-resistant channel or key exchange protocols such as Diffie–Hellman.
ii. When a new secret S_k is to be deployed, it is encoded to E_k by key $K_k = g^{af}$.
iii. Then (12) is used to distribute f by placing it at the free coefficient of the polynomial;
iv. The other steps of Algorithm 1 remain the same, except for using modular exponentiation instead of PUF to compute K_k at the user end.

Note: Except that the key K_k is updated in a one-way manner, the mechanism above is similar to the double ratchet algorithm [26], known as the 'silent whisper' used in secure messaging apps nowadays.

Algorithm 3: Another approach is to leverage the seeded pseudo random number generators (PRNGs). Denote the PRNG function as PRNG(seed).

i. The dealer and the client firstly achieve an agreement on a secret r_0. This can also be implemented by a secure channel or key exchange protocols.
ii. When a new secret S_k is to be deployed, it is encoded to E_k by K_k, where $K_k = \text{PRNG}(r_0 + r1)$.
iii. Equation (12) is used to distribute r_1 by placing it at the free coefficient of the polynomial.
iv. The other steps of Algorithm 1 remain the same, except for using PRNG() instead of PUF to compute K_k at the user end.

Note: The '+' here stands for a combination of r_0 and r_1. It can be operations such as bitwise XOR, concatenation, or modular addition etc.

Remark 3: In terms of security, the three EtM key update algorithms all provide similar trustworthiness in the transmission of the EtM key. They do not need a specific secure channel for the transmission since the protocols are already eavesdrop resistant. The PUF-based approach adopted in this work has two key advantages: (i) unlike the key exchange-based using modular exponentiation and the PRNG-based algorithms, Algorithm 1 has its security rooted in the hardware uniqueness of the PUF, which is hard to forge or duplicate unless the adversary physically acquires the device; and (ii) The PUF-enabled design is the most cost-efficient, in terms of hardware area, when the RASSS protocol is applied to physical devices. This point is captured in the implementation results shown in Table 1. Key exchange-based and PRNG-based schemes both need to take extra circuit and computation resources (cf. Section 6.4).

5.4 Cheater identification by group testing

In Section 3.2.2, we pointed it that the secret verification alone does not identify the cheaters nor help to retrieve the correct secret. Therefore in this study, we propose an adaptive group testing which works together with secret verification for cheater identification. It can locate up to $c_{\text{est}} = n - t$ number of cheaters, which is the theoretical upper bound. Meaning in a t-TSS scheme, among all the n shareholders participating in our scheme, our scheme only needs as few as only t honest parties to retrieve the correct secret.

There are two group testing stages, a small-scale testing stage and a large-scale. The former is based on superimposed codes with limited cheater identification, and the latter does an exhaustive search to locate up to $n - t$ cheaters. The superimposed code is defined as follows.

Definition 5: Let $M_{i,j} \in \{0, 1\}$ be the element in row i and column j in a binary matrix $\boldsymbol{M}$ of size $A \times N$. The set of columns of $\boldsymbol{M}$ is called a c-superimposed code, if for any set W of up to c columns and any single column $h \notin W$, there exists a row f in the matrix $\boldsymbol{M}$, for which $\boldsymbol{M}_{f,h} = 1$ for column f, and $\boldsymbol{M}_{f,j} = 0$ for all $j \in W$ [27, 28]. The above property is called zero-false-drop of order c. It follows that for all the N columns in $\boldsymbol{M}$, the Boolean OR of up to any c columns are all different.

When $\boldsymbol{M}$ is used as the group testing pattern to identify the cheaters, the 1's in each row (test) correspond to the shareholders participating in that particular test. Each test is a two-step procedure:

i. A secret reconstruction using (2) to retrieve the secret $\tilde{E}$ with its specific participants.
ii. An authentication using (7) or (9) over $\tilde{E}$ to verify the validity of the retrieved secret.

The returned test syndrome is a T-bit binary vector $\boldsymbol{u}$, where 0's in $\boldsymbol{u}$ indicate the equality of (7) or (9), and 1's the inequality. For up to c cheaters, each cheater combination will have a different test syndrome.

For example, the columns of the following matrix are c-superimposed code, where $c = 1$:

$$\boldsymbol{M} = \begin{vmatrix} 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 \end{vmatrix}.$$

Using this test pattern will enable a client to locate one cheater out of five devices within four tests. If device $D_2 = 2$ is the cheater, then its test syndrome will uniquely be $\{1, 1, 1, 1\}$.

When $c > 1$ the construction of a c-superimposed code matrix is no longer trivial. Here we suggest the construction based on error correction codes.

Construction 1: Let C_q be a $(m_q, k_q, d_q)_q$ q-ary ($q = p^s$ is a power of prime and $p \neq 2$) conventional error correcting code. Each digit of C_q in GF(q) is represented by a q-bit binary vector with Hamming weight one. A superimposed code C_{SI} can be constructed by substituting every q-ary digit of codewords in C_q by its corresponding binary vector. The resulted $A \times N$ matrix $\boldsymbol{M}$ of the c-superimposed code C_{SI} has the following parameters [29]:

$$\begin{aligned} A &= qm_q; \\ N &= q^{k_q}; \\ c &= \left\lfloor \frac{m_q - 1}{m_q - d_q} \right\rfloor. \end{aligned} \tag{14}$$

If C_q is a MDS q-nary code, for which $d_q = m_q - k_q + 1$, such as RS codes, then c can be written as

$$c = \left\lfloor \frac{m_q - 1}{k_q - 1} \right\rfloor, \tag{15}$$

where c is the cheater identification capability in this construction.

The selection of this construction is for the reason that, for the $A \times N$ matrix $\boldsymbol{M}$, in every row, there is exactly the same number of 1's. This feature is suitable for the use of TSS secret retrieval since every retrieval requires the same number of devices.

However, the group testing pattern constructed above has a very limited cheater identification capability, as shown in (15). When there is a large number of cheaters, we propose the following large-scale group testing scheme.

Construction 2: For any t-TSS scheme, suppose among n holders there are c attackers where $0 \leq c \leq n - t$. A test pattern to identify the honest holders and attackers can be constructed as a binary matrix $\boldsymbol{M}$ of size $T \times n$, where T is the number of tests needed at most. The rows of $\boldsymbol{M}$ consist of all different n-bit vectors with exactly t 1's and so $T = \binom{n}{t}$. Each column of the matrix, therefore, has $\binom{n-1}{t-1}$ number of 1's. The 1's in each row (test) correspond to the shareholders participating in that particular test. Each test is a two-step procedure:

i. A secret reconstruction using (2) to retrieve the secret $\tilde{E}$ with its specific participants.
ii. An authentication using (7) or (9) over $\tilde{E}$ to verify the validity of the retrieved secret.

The test syndrome is a T-bit binary vector $\boldsymbol{u}$, where 0's in $\boldsymbol{u}$ indicate the equality of (7) or (9), and 1's the inequality.

Then the cheater identification algorithm is

Algorithm 4: For any t-TSS scheme and its corresponding group testing matrix $\boldsymbol{M}$ there are n shareholders participating in the tests indexed by $H = \{0, 1, 2, \ldots, n-1\}$. Among the n shareholders, there are c_{est} cheaters where $n/3 \leq c_{\mathrm{est}} \leq n - t$. Let $\boldsymbol{w} = (w_0, w_1, \ldots, w_{n-1})$ be a n-digit vector and $w = u^{\mathrm{T}} \times M$, where u is the T-bit binary test syndrome and $\times$ is the multiplication of regular arithmetic. The cheaters' indices belong to the set $\left\{ l \middle| w_l = \binom{n-1}{t-1} \right\}$. and the rest of the holders are honest.

However, the testing technique above requires $\binom{n}{t}$ tests in total to identify the cheaters. This can be a large number when n and t are large. Therefore its adaptive form is given below which drastically reduces the average number of tests to a linear formula.

Algorithm 5: For a test pattern $\boldsymbol{M}$ of size $T \times n$ generated by Construction 2, ΔT is the number of tests needed to find the first 0 (equality of (7) or (9)) in the test syndrome. The n shareholders are indexed by $H = \{0, 1, 2, \ldots, n-1\}$. The t honest holders identified by this test are indexed by $I = \{i_0, i_1, \ldots, i_{t-1}\}$. The system only needs to run at most $n - t$ more tests whose participants are $\{i_0, i_1, \ldots, i_{t-2}, j\}$, where $j \in H \setminus I$. Each test's syndrome indicates holder j as an attacker or not by 1 or 0. The total number of tests needed to identify all holders is then at most $\Delta T + (n - t)$.

5.5 Extra invitation module

If the group testing module in Stage 4 cannot successfully identify the c_{act} cheaters in the system, where $n - t < c_{\mathrm{act}} \leq n$, then the number of honest shareholders is less than t.

At this point, our scheme will still raise the cheating alarm based on the secret authentication. Moreover, the protocol is adaptive enough to be extended to a further stage to include an invitation module. This module can pull in additional participants and perform new rounds of group testing. From the hardware perspective, the invitation module can be power-gated and disabled when not in use.

Algorithm 6: Let the number of honest shareholders in the current group testing be Δt and $0 \leq \Delta t < t$. Suppose the system is able to identify an extra set of t honest shareholders from another group. Then these t honest parties can be combined into the current group with the modified group testing matrix of size $\binom{n+t}{t} \times (n + t)$. With this new test pattern, the $\Delta t + t$ honest shareholders can be identified and the rest will be properly labelled as cheaters.

5.6 Numeric examples

Here we present two illustrative examples to demonstrate the security of the proposed protocol. The first one will be under the passive attack and the second one under the active attack.

Example 3: For an Odysseus system equipped with the proposed secret sharing scheme, there are t cheaters who want to stealthily compute the original secret S. However, what they can acquire are $E = \mathrm{EtM}(K, S)$ and CHL. Without the client's PUF they are not able to have the response K to CHL. Therefore, S still remains unknown to the t curious cheaters.

For the second example, for simplicity, we will not perform the encryption function ENC() in the EtM. For the MAC function, we will use AMD() since it is able to work with very short vectors. Thus this numeric example will be relatively small and easy to follow.

Example 4: In an Odysseus system, there are seven boards distributed. This system has adopted our proposed secure TSS scheme which is t-threshold and $t = 3$. The original secret is a digital signature $S \in \mathrm{GF}(2^{12})$, where $S = 001111110000 = 0x3F0$. The RS decoder in this scheme is constructed under the assumption that there are at most two cheaters. However, in this scenario, there are four devices which have been compromised by the cheaters.

Stage 1: secret encoding and share distribution: The original secret $0x3F0$ is first encoded by the AMD encoding equation (8). Using Definition 3, we choose $b = 4$ such that the encoding and decoding are over GF(2^4), $m = 1$ such that the random vector has only one symbol and $g = 3$ such that S is partitioned into three symbols $S = (S_0, S_1, S_2)$ where $S_0 = 0x3$, $S_1 = 0xF$, and $S_2 = 0x0$. Suppose the dealer has chosen a response from the client's PUF which is $K = 0x0006$ whose corresponding challenge is $\mathrm{CHL} = 0xAAAA$. The original secret will be encoded to an AMD codeword $E = \mathrm{AMD}(K, S)$ by

$$\mathrm{AMD}(K, S) = S_0 K \oplus S_1 K^2 \oplus S_2 K^3 = 0x1 \Rightarrow E = (0x3F01).$$

Then with the share distribution polynomial

$$h_i = \mathrm{CHL} \oplus a_1 D_i \oplus \mathrm{ED}_i^2,$$

where $a_1 = 0x5555$ is an arbitrarily chosen coefficient and $\mathrm{CHL}, a_1, E \in \mathrm{GF}(2^{16})$, this encoded secret is shared to seven Odysseus boards with IDs and shares $\{D_i{:}h_i\} = \{1{:}0xC0FE\}$, $\{2{:}0xFC04\}$, $\{3{:}0x9650\}$, $\{4{:}0x0FB4\}$, $\{5{:}0x65E0\}$, $\{6{:}0x591A\}$, and $\{7{:}0x334E\}$.

However, devices $\{3, 4, 6, 7\}$ have been compromised by cheaters and they have collusively selected another secret $\tilde{S} = 0xABCD$ and forged another share distribution polynomial:

$$\tilde{h}_i = 0xAAAA \oplus 0x7777 D_i \oplus 0xABCD \cdot D_i^2.$$

By their IDs, their shares are changed to $\{3{:}0x2686\}$, $\{4{:}0xDBAF\}$, $\{6{:}0x9A2F\}$, and $\{7{:}0x4695\}$.

Stage 2: secret reconstruction and verification: suppose firstly Odysseus devices $\{2, 3, 4\}$ are selected to reconstruct the secret with $\{3, 4\}$ being cheaters. By the secret reconstruction (2) the retrieved secret is

$$\tilde{E} = 0x5522.$$

The reconstructed secret will be verified by the AMD decoder using (9) $\overline{\mathrm{AMD}(K,S)} \stackrel{?}{=} \mathrm{AMD}(\tilde{K},\tilde{S})$. Through the computation over $\mathrm{GF}(2^4)$ we have the following inequality:

$$\overline{\mathrm{AMD}(K,S)} \neq [\mathrm{AMD}(\tilde{K},\tilde{S}) = \tilde{S}_0 \tilde{K} \oplus \tilde{S}_1 \tilde{K}^2 \oplus \tilde{S}_2 \tilde{K}^3].$$

Thus, cheating is detected and Stage 3 will be initiated under the assumption of $c_{\mathrm{est}} = 2$ cheaters.

Stage 3: share error correction: Under the RS decoder, $n = 3c_{\mathrm{est}} + 1 = 7$ shareholders will be involved and it can correct up to two shares using an $(n, t, d) = (7, 3, 5)$ RS code. However, there is a total number of $c_{\mathrm{act}} = 4$ cheaters $\{3, 4, 6, 7\}$ which is beyond the capability of this RS decoder. Therefore, the protocol moves in its fourth stage upon the failure of error correction.

Stage 4: small-scale group testing: A 9×9 group testing matrix is generated based on a two-superimposed code by using a $(3, 2, 2)_3$ ECC code through construction 1. By truncating two columns it can be used to locate two cheating Odysseus boards among seven in nine tests Fig. 8.

However, this group test is still not able to identify the cheaters, since $c_{\mathrm{act}} = 4 > 2$. Thus Stage 5 has to be activated.

Stage 5: large-scale group testing: This stage is designed under the assumption that among all the seven Odysseus boards from Stage 3, only $t = 3$ are not compromised by cheaters. The group testing matrix $\boldsymbol{M}$ of size $T \times n$ can be constructed with Construction 5.1, where $T = \binom{n}{t} = 35, n = 7$. To save space $\boldsymbol{M}$ is listed transposed as $\boldsymbol{M}^{\mathrm{T}}$ Fig. 9:

Each test involves three boards and the secret retrieved by them is to be verified by (9). Since boards $\{1, 2, 5\}$ are not compromised by cheaters, test 7 is the first test with syndrome 0.

Based on the adaptive Algorithm 5.3, $\Delta T = 7$. The system will only need to run the tests of $\{1, 6, 8, 9\}$ whose participants are boards $\{1, 2, j\}$ where $j \in H \setminus I = \{3, 4, 6, 7\}$. Thus only tests $\{8, 9\}$ are left to run. The actual number of implemented tests are then $9 < \Delta T + (n - k) \ll \binom{n}{k} = 35$.

In this way, the Odysseus boards which have been hijacked by cheaters are identified as $\{3, 4, 6, 7\}$. Also, the properly functional boards $\{1, 2, 5\}$ will be able to retrieve the encoded legal secret $E = 0x3F01$ and therefore the correct digital signature is $S = 0x3F0$.

6 Design evaluation and automation

In this section, we will evaluate the proposed scheme and offer a design automation tool for it.

	1	2	3	4	5	6	7	8	9
1	1	0	0	1	0	0	1	0	0
2	1	0	0	0	1	0	0	0	1
3	1	0	0	0	0	1	0	1	0
4	0	1	0	1	0	0	0	0	1
5	0	1	0	0	1	0	0	1	0
6	0	1	0	0	0	1	1	0	0
7	0	0	1	1	0	0	0	1	0
8	0	0	1	0	1	0	1	0	0
9	0	0	1	0	0	1	0	0	1

Fig. 8 *The test pattern generated assuming up to 4 cheaters are involved*

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35
1	1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	1	0	0	0	1	1	1	1	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0
3	1	1	1	0	0	0	0	0	0	1	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	1	1	0	0
4	0	1	1	1	0	1	0	0	0	1	0	0	0	1	1	1	0	0	0	0	0	0	1	1	1	0	0	0	1	1	0	0	0	1	1
5	0	0	1	1	1	0	1	0	0	0	1	0	0	1	0	0	1	1	0	1	0	0	1	0	0	1	1	0	0	0	1	1	0	1	0
6	0	0	0	1	1	0	0	1	0	0	0	1	0	0	1	0	1	0	1	0	1	0	0	1	0	1	0	1	1	0	1	0	1	0	1
7	0	0	0	0	1	0	0	0	1	0	0	0	1	0	0	1	0	1	1	0	0	1	0	0	1	0	1	1	0	1	0	1	1	1	1

Fig. 9 *The test pattern generated assuming no more than 2 cheaters are involved*

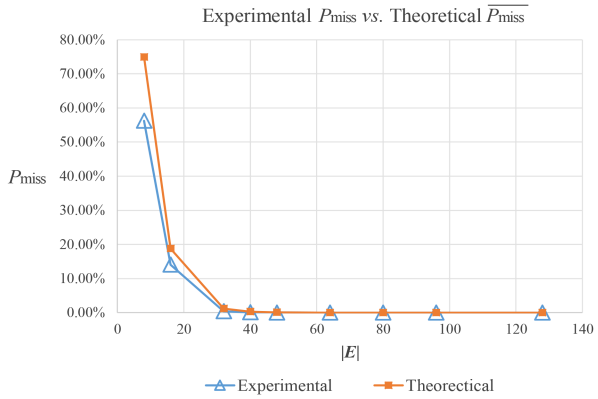

Fig. 10 *Experimental P_{miss} matches the theoretical upper bound $\overline{P_{miss}} = (g/2^b)$. The experimental results are usually better than the upper bound because (9) does not always have h solutions in the finite field. Also, when $b \geq 32$ the experiments did not miss a single attack*

6.1 Design evaluation

In the previous example, the AMD code works over $\mathrm{GF}(2^4)$, where the error mis-detection probability is $\overline{P_{\mathrm{miss}}} = (3/2^4)$ in the worst case. To increase the security level, one can simply have the protocol work over a larger field. If the system uses HMAC as the MAC() function, then P_{miss} is a fixed value. Therefore, we will only test the performance of the AMD() under different vector sizes.

In our experiments, the sizes of the encoded secret E are set to $\{8, 16, 32, 48, 64, 80, 96, 128\}$ bits, which are the cases for most real-world applications. Therefore, the AMD codes are over $\mathrm{GF}(2^b)$ fields where $b \in \{2, 4, 8, 12, 16, 20, 24, 32\}$. A comparison is made between the experimental P_{miss} (under $4 \cdot 2^b$ rounds of attack and defence) and the theoretical $\overline{P_{\mathrm{miss}}}$ (Fig. 10).

6.2 Hardware and timing overhead

The hardware cost comparison between RASSS and the classic scheme (original secret sharing with share verification, c.f. section 3) in Table 2 is made on a Xilinx Vertex 7 XC7VX330T FPGA board under the same parameters as in Section 6.1.

The timing comparison is made under a severely adverse scenario with $c_{\mathrm{act}} \leq n - t$ cheaters for RASSS, and much fewer cheaters of $c_{\mathrm{act}} \leq n/3 - 1$ for the classic scheme. It is implemented by Python on an Intel® Core™ i7-6700 @ 3.4 GHz and 8 GB memory machine running Linux operating system.

Table 2 Hardware and timing overhead

E (bits)	Hardware (slices)			Timing (10^6 clock cycles)		
	Classic	RASSS	Ratio	Classic	RASSS	Ratio
8	521	828	0.59	0.47	3.80	7.09
16	1492	2256	0.51	0.56	5.62	9.04
32	3977	6164	0.55	1.36	16.24	10.94
48	6114	9462	0.55	1.89	24.68	12.06
64	8462	12,749	0.51	2.55	29.90	10.73
80	9895	15,804	0.59	3.18	36.92	10.61
96	11,873	18,918	0.59	3.68	45.05	11.24
128	17,842	27,695	0.55	4.79	58.55	11.22

[a]Ratio = (RASSS/Classic) − 1.

[b]With only 60% of the hardware overhead the RASSS protocol drastically improves the cheater tolerance capability. The latency of the classic protocol is 49 logic steps and the latency of RASSS 223 steps.

[c]Although the RASSS protocol has a large T as an upper bound, with the adaptive test in Algorithm 5 it effectively reduces the actual number of tests.

Table 3 Cheater tolerance in the Odysseus system

n	$t = 2$		$t = 3$		$t = 4$	
	Classic	RASSS	Classic	RASSS	Classic	RASSS
10	3	8	3	7	3	6
20	6	18	6	17	6	16
30	9	28	9	27	9	26
40	13	38	13	37	13	36
50	16	48	16	47	16	46

[a]Different numbers of cheaters are injected in the tests – where for an Odysseus network of size n and secret sharing threshold t, we have $1 \leq c_{\text{act}} \leq n - t$. The results recorded in the table are the maximum number of cheaters that a particular scheme can tolerate.

[b]In almost all the tested Odysseus system settings, the proposed RASSS protocol consistently provides better cheater tolerance than the classic scheme, especially when n is large. For example, when $n = 50$, RASSS is able to (i) identify and tolerate 30 more cheaters than the classic scheme, and (ii) retrieve the correct secret.

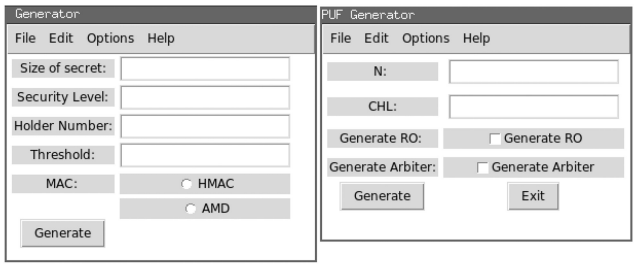

Fig. 11 *Graphical user interfaces for the secret sharing system generator (left) and the PUF generator (right). With this tool, any researcher/user interested in PUF-based secret sharing can easily generate their own customised secret sharing system in just a few clicks. We plan to release this automation instrument as an open-source design tool*

6.3 Cheater detection and identification in the Odysseus system

In a simulation, an $E \geq 96$-bit version of the proposed RASSS protocol achieves near perfect cheater tolerance. In 17 billion tests, almost no attacks are mis-detected. To evaluate the robustness and scalability of the protocol under practical settings, we implement and deploy an $E = 128$-bit version of the protocol on the Odysseus platform (cf. Section 2). Five Odysseus system network sizes are tested. In other words, we have $n \in \{10, 20, 30, 40, 50\}$, where n is the number of edge nodes (sensor boards) in the system. Each of the five systems is configured with three thresholds: $t \in \{2, 3, 4\}$ in the secret sharing scheme. In total, the cheater tolerance tests, therefore, are conducted under 15 different Odysseus system setups. Similar to Table 2, a comparison is made in Table 3 between the classic cheater tolerance scheme (with share verification by RS decoder) and RASSS.

It should be noted that although the RASSS protocol guarantees a stronger cheater tolerance, under the Odysseus system testing, it also incurs a longer convergence delay. As shown in Table 2, for $E = 128$-bit version of the RASSS protocol, the delay is about 11 times that of the classic scheme. This delay is due to a large number of group testings in Stage 5 of the protocol (cf. Section 5.1). Therefore, when deploying the protocol in large network applications such as the Odysseus network, the group testing module (which involves the test matrix) should be adjusted in order to achieve the desirable security-cost/delay trade-off.

6.4 Encrypt-then-MAC (EtM) key transmission primitives

In Sections 5.3.2 and 5.3.3, we introduced three approaches to performing the EtM keys transmission. As mentioned in Remark 3, all three techniques provide the same level of security to the key transmission process. However, their implementations do require different hardware complexities and costs. Table 1 summaries the hardware resources utilisation for their FPGA-based implementations, as well as the major operations (Major Op) used in each approach.

When the RASSS protocol is deployed in IoT systems with limited hardware resources and small form factor edge nodes, such as the Odysseus edge nodes (cf. Section 2), the selection of the EtM key update algorithm is important. The PUF-based technique lends itself particularly well to such settings.

6.5 Design automation

Although one can manually make a secret sharing system with PUF on FPGAs, it still involves a good amount of work: writing the hardware description language (HDL) code, fixing the routing and placement of PUF's basic elements, and configuring the bitstream etc. Also, with a change of a parameter, the entire system may need to be modified. Therefore, we have designed an automation tool (Fig. 11), which simply takes user's inputs of four/five parameters: secret size, security level (for AMD only, HMAC default as 2^{-128}), total number of holders n, threshold t, and MAC function. In addition, we also provide a PUF automation tool to generate the PUFs based on user specified response and challenge sizes.

In this tool, the system's HDL codes and PUF's fixed-routing configuration are pre-written in a folder named 'Templates'. The tool will generate the system according to user specified parameters based on the files in this folder. In future updates to the

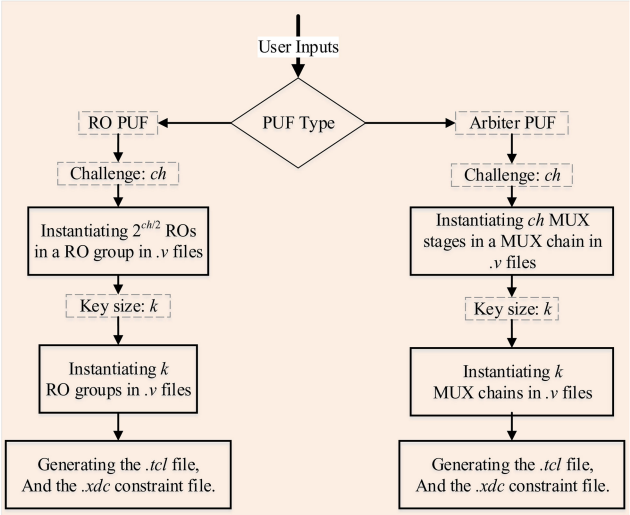

Fig. 12 *This tool is able to generate .v Verilog HDL source files, .tcl bitstream configuration files, and a .xdc constraint file, with which a researcher can request his/her customised PUF. However, the input and output ports need to be defined by the researcher in the .v top module file since different FPGAs have different ports*

tool, the templates could be modified while the generator tool remains unchanged.

Although the RASSS scheme automatic generation follows Section 5.1 and enough details have been given, the design automation for the PUF has not been fully discussed, due to the focus and content limitation of the study. Thus, in Fig. 12, we provide the PUF automation's work flow, which generates two different types of PUFs (RO and Arbiter PUF [30]) with the user defined challenge and response (key) sizes.

7 Conclusion

In this study, we have proposed a secure and robust scheme to share confidential information in distributed systems. This scheme uses TSS to split the information into shares to be kept by all devices in the system. Thus the malfunction of a single device will not harm the security of the entire system. Given a larger number of malfunctional devices, the scheme ensures both the privacy and integrity of that piece of information even when there is a large amount of sophisticated and collusive attackers who have hijacked the devices. Furthermore, it is able to identify all the compromised devices, while still keeping the secret unknown and unforgeable to attackers. Overall, this scheme works in an adaptive manner; a more powerful and resource consuming security module will only be activated when the previous modules fail. Therefore the average power consumption is minimised. In terms of application, this scheme can be applied but not limited to most of the IoT systems with a structure similar to the Odysseus.

8 Acknowledgments

This research was partially supported by the NSF grant (no. CNS-1745808).

9 References

[1] Khandelwal, S.: 'Millions of IoT devices using same hard-coded CRYPTO keys', 2015. Available at http://thehackernews.com

[2] Chadha, A., Liu, Y., Das, S.K.: 'Group key distribution via local collaboration in wireless sensor networks'. IEEE SECON Proc., Santa Clara, CA, USA, 2005

[3] Geambasu, R., Kohno, T., Levy, A.A., *et al.*: 'Vanish: increasing data privacy with self-destructing data'. USENIX Security Symp., Montreal, Canada, 2009

[4] http://go.thalesesecurity.com/rs/480-LWA-970/images/Microsoft-AD-CS-and-OCSP-Integration-Guide-for-Microsoft-Windows-Server-2016.pdf, accessed May 2017

[5] Able, J., *et al.*: 'DNSSEC root zone high level technical architecture', 2010. Available at http://www.rootdnssec.org/wp-content/uploads/2010/06/draft-icanndnssec-arch-v1dot4.pdf

[6] Lapets, A., Volgushev, N., Bestavros, A., *et al.*: 'Secure multi-party computation for analytics deployed as a lightweight web application', Science Department, Boston University, 2016

[7] Bu, L., Nguyen, H., Kinsy, M.A.: 'RASSS: a perfidy-aware protocol for designing trustworthy distributed systems'. Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Cambridge, UK, 2017

[8] Shamir, A.: 'How to share a secret', *Commun. ACM*, 1979, **22**, (11), pp. 612–613

[9] McEliece, R.J., Sarwate, D.V.: 'On sharing secrets and Reed–Solomon codes', *Commun. ACM*, 1981, **25**, (9), pp. 583–584

[10] Gennaro, R., Ishai, Y., Kushilevitz, E., *et al.*: 'The round complexity of verifiable secret sharing and secure multicast'. 33rd Annual ACM Symp. on Theory of Computing, Crete, Greece, 2001

[11] Fitzi, M., Garay, J., Gollakota, S., *et al.*: 'Round-optimal and efficient verifiable secret sharing'. Theory of Cryptography Conf., New York, NY, USA, 2006

[12] Berlekamp, E.: '*Algebraic coding theory: revised edition*' (World Scientific, Singapore, 2015)

[13] Gao, S.: 'A new algorithm for decoding Reed-Solomon codes', '*Communications, information and network security*' (Springer, Boston, MA, USA, 2003), pp. 55–68

[14] Krawczyk, H.: 'Secret sharing made short'. Annual Int. Cryptology Conf., Santa Barbara, CA, USA, 1993

[15] Liu, J., Mesnager, S., Chen, L.: 'Secret sharing schemes with general access structures'. Int. Conf. on Information Security and Cryptology, Seoul, Republic of Korea, 2015

[16] Krawczyk, H., Canetti, R., Bellare, M.: 'HMAC: keyed hashing for message authentication', RFC2104, 1997

[17] Eastlake 3rd, D., Hansen, T.: 'US secure hash algorithms (SHA and SHA-based HMAC and HKDF', RFC6234, 2011

[18] Cramer, R., Dodis, Y., Fehr, S., *et al.*: 'Detection of algebraic manipulation with applications to robust secret sharing and fuzzy extractors'. Annual Int. Conf. on the Theory and Applications of Cryptographic Techniques, Istanbul, Turkey, 2008

[19] Wang, Z., Karpovsky, M.G.: 'Algebraic manipulation detection codes and their applications for design of secure cryptographic devices'. IEEE On-Line Testing Symp., Athens, Greece, 2011

[20] Bu, L., Karpovsky, M.G.: 'A design of secure and reliable wireless transmission channel for implantable medical devices'. 3rd Int. Conf. on Information Systems Security and Privacy, Porto, Portugal, 2017

[21] Bauder, D.: 'An anti-counterfeiting concept for currency systems'. Research report PTK-11990, Sandia National Labs, 1983

[22] Gassend, B., Clarke, D., Van Dijk, M., *et al.*: 'Silicon physical random functions'. Proc. Computer and Communications Security Conf., Washington, DC, USA, 2002

[23] Yu, M.-D., Devadas, S.: 'Pervasive, dynamic authentication of physical items', *Queue*, 2016, **60**, (4), pp. 32–39

[24] https://www.eetimes.com/document.asp?doc_id=1254886, accessed February 2018

[25] Morozov, S., Maiti, A., Schaumont, P.: 'An analysis of delay based PUF implementations on FPGA'. Int. Symp. on Applied Reconfigurable Computing (ARC), Bangkok, Thailand, 2010

[26] Marlinspike, M., Perrin, T.: 'The double ratchet algorithm.(2016)', 2016. Available at https://whispersystems.org/docs/specifications/doubleratchet

[27] D'yachkov, A.G., Macula, A.J., Rykov, V.V.: 'On optimal parameters of a class of superimposed codes and designs'. IEEE Int. Symp. on Information Theory, Cambridge, MA, USA, 1998

[28] D'yachkov, A.G., Macula, A.J., Rykov, V.V.: 'New applications and results of superimposed code theory arising from the potentialities of molecular biology', '*Numbers, information and complexity*' (Springer, Boston, MA, USA, 2000), pp. 265–282

[29] Bu, L., Karpovsky, M., Wang, Z.: 'New byte error correcting codes with simple decoding for reliable cache design'. IEEE On-Line Testing Symp. (IOLTS), Halkidiki, Greece, 2015

[30] Herder, C., Yu, M.-D., Koushanfar, F., *et al.*: 'Physical unclonable functions and applications: a tutorial', *Proc. IEEE., Piscataway, NJ, USA*, 2014, **102**, (8), pp. 1126–1141