applied sciences

MDPI

Article

Towards Secure and Usable Certificate-Based Authentication System Using a Secondary Device for an Industrial Internet of Things

Jusop Choi [1], Junsung Cho [1], Hyoungshick Kim [1,2] and Sangwon Hyun [3,*]

[1] Department of Computer Science and Engineering, Sungkyunkwan University, Suwon, Gyeonggi-do 16419, Korea; cjs1992@skku.edu (J.C.); js.cho@skku.edu (J.C.); hyoung@skku.edu (H.K.)
[2] Data61, CSIRO, Marsfield, NSW 2122, Australia
[3] Department of Computer Engineering, Myongji University, Yongin, Gyeonggi-do 17058, Korea
* Correspondence: shyun@mju.ac.kr

Received: 21 February 2020; Accepted: 11 March 2020; Published: 13 March 2020

Abstract: As the number of controllers and devices increases in Industrial Internet of Things (IIoT) applications, it is essential to provide a secure and usable user authentication system for human operators who have to manage tens or hundreds of controllers and devices with his/her password. In this paper, we propose a *formally verified* certificate-based authentication system using a secondary network device for such IIoT applications. In the proposed system, a user's sign key is encrypted with a secret key that can be computed with his/her password and a secret parameter in a secondary device to securely protect the sign key. To demonstrate the feasibility of the proposed system, we implemented a prototype with standard cryptographic algorithms (AES-256, RSA-3072, and ECDSA-256). The experiment results demonstrated that the execution time overhead of the sign key recovery process was 0.039 and 0.073 s, respectively, for RSA-3072 and ECDSA-256, which was marginal compared with the total execution time (0.383 s for RSA-3072 and 0.319 s for ECDSA-256) of the conventional system. We also verified the security of the proposed protocol using a formal verification tool called ProVerif.

Keywords: industrial controller; key management; key protection; user authentication

1. Introduction

One of the key requirements for Industrial Internet of Things (IIoT) applications is to establish trust among things (e.g., industrial controllers, sensors, and actuators) and human operators [1]. Interestingly, secure authentication protocols using cryptographic primitives are generally used for things, whereas passwords are still popularly used for human operators. This is natural—human operators cannot remember random cryptographic keys with high entropy.

In IIoT applications, human operators often manage and monitor tens or hundreds of controllers. Therefore, it is inevitably required for them to create and remember passwords to securely access those controllers. This authentication burden often leads to bad security practices, such as using a weak (easy-to-remember) password [2]. Periodically changing passwords (e.g., every three months) is a further large burden as an operator must access every controller individually to change their passwords. Moreover, some controllers provide very limited ways of physically entering text (e.g., up and down buttons); and operators are expected to scroll up and down a given list of characters to select their passwords. Such limitations on the physical user interface make it difficult and inefficient for operators to create, update, and enter passwords.

To overcome the usability and management issues of passwords and authenticate operators in a secure and user-friendly manner, a certificate-based authentication protocol can also be considered

for human operators. Basically, a user authentication process can briefly be summarized as follows. Suppose that a human operator's sign key is installed on the operator's client device such as a smartphone or smartwatch, where a client application for the user authentication process is installed. To communicate with a controller, a client application (instead of the operator) first obtains the sign key, computes an authenticated code with the sign key, and sends the authenticated code to the controller to prove the ownership of the sign key. The controller verifies the authenticated code received from the client with the corresponding verification key. We note that key protection is very important to ensure the security of the authentication protocol. However, most client devices do not have hardware support for protecting the user's sign key. Again, passwords are still popularly used to encrypt the sign key (e.g., PBKDF2 [3]) and store only the encrypted sign key on the client device. This encryption key (used to encrypt the user's sign key) should then only be recovered with the user password. In this case, the security of the key protection mechanism depends on the strength of the user's password: an attacker who has access to the encrypted sign key stored in the client device would be able to perform an offline dictionary- and rule-based brute-force attack to guess the user's password and then decrypt the sign key. If the attacker can obtain the user's sign key, they can connect to all of the controllers the user has access to.

In this paper, we present a certificate-based user authentication system where a user's sign key is encrypted with a key derived from both the user password and a specific secret value, without relying on special hardware. The proposed authentication method makes use of a secondary device (e.g., an honest-but-curious server or smartphone) that the user can connect to for protecting the user's sign key. Our key protection idea works by storing the specific secret value, which is used to generate the encryption key, in a secondary device—making it infeasible for attackers to perform offline dictionary attacks even when the primary client device is stolen. To perform an offline dictionary attack against the proposed scheme, an attacker would have to break into both the primary and secondary devices at the same time to steal the encrypted sign key, as well as the specific secret value. We argue that such an attack would be computationally very expensive and difficult to perform. Our key contributions can be summarized as follows:

- We propose a *formally verified* certificate-based user authentication method that does not require additional secure hardware (e.g., ARM TrustZone) to mitigate offline password guessing attacks. The user's sign key is encrypted and the encrypted key can only be decrypted when the primary device successfully communicates with a network-connected secondary device that assists in the decryption process. We proved that the proposed protocol is secure using `ProVerif` that is an automatic cryptographic protocol verifier in the formal model.
- We demonstrate the feasibility of the proposed certificate-based authentication method by implementing a prototype with popularly used cryptographic algorithms (AES-256, RSA-3072, and ECDSA-256). With this prototype, we analyze its communication overhead and demonstrate that the execution time overhead of the signing key recovery process is acceptable (0.039 s for RSA-3072, 0.073 s for ECDSA-256 on average compared with the total execution time of the conventional system) for the industrial controller application.

The rest of this paper is organized as follows. In Section 2, we provide an overview of the related work. In Section 3, we explain the conventional authentication method with a digital certificate. In Section 4, we describe possible attack scenarios against the authentication method. In Section 5, we present a certificate-based user authentication method using a secondary device. In Section 6, we analyze the security of the proposed system using a well-known verification tool called ProVerif. In Section 7, we evaluate the performance of our prototype implementation. Finally, the conclusions of the study are provided in Section 8.

2. Related Work

Local storage protection has been actively studied in academia. Most issues relating to the encryption of local data have been addressed using a key that is derived from a user password [3]. Canetti et al. [4] proposed an approach for limiting offline dictionary attacks by additionally requiring the user to solve a puzzle, such as CAPTCHA during the process of deriving a key from the user password. Catuogno et al. [5] proposed a secret key sharing scheme in which a user's master key is reconstructed with two partial secrets that are stored and managed, respectively, in two separate devices. At first glance, this approach seems similar to ours in the sense that a user's master key has to be obtained only through the cooperation of multiple devices. However, the proposed system requires a centralized administration entity's persistent intervention and management, whereas in our approach the centralized administration entity is only needed for initial access policy configurations for industrial controllers. In addition, they presented the design of a system only without the actual implementation of their system while we implemented a fully working system and evaluated its performance under various conditions. Catuogno et al. [6] proposed a secure storage architecture based on the TrustZone technology [7] in mobile devices to protect sensitive files from unauthorized access. However, the proposed technique can only be used with TrustZone enabled devices. One of our design goals is to provide a secure certificate-based user authentication scheme without relying on additional special hardware components such as TrustZone. To achieve this goal, we develop a secure user authentication scheme and evaluate its efficiency and security through experiments under various conditions. Unlike previous studies, we formally verified the security of our system.

The National Institute of Standards and Technology (NIST) published a document for securing Industrial Control Systems (ICS) [8]. This document analyzes potential threats and vulnerabilities to ICS and recommends suitable countermeasures against the identified threats. In particular, with respect to user authentication, this document discusses five methods that could be applied to ICS environments and their pros and cons: password, challenge/response, physical token, smart card, and biometric authentication. This document also discusses specific considerations of ICS when applying each of these authentication methods. Borisov et al. [9] proposed a user authentication scheme based on QR codes and one-time passwords for industrial control systems, and Plaga et al. [10] further improved the scheme in [9] by taking advantage of the TLS channel binding technique [11]. Huh et al. [1] proposed generic system architecture for user authentication and access control to industrial controllers in distributed industrial control systems. In their proposed architecture, a centralized authentication server and a centralized policy management server are employed for more efficient and flexible access control. Despite many benefits provided by such architectures with centralized servers, centralized identity management systems can be attractive targets for attackers, becoming a single point of failure.

Abidin et al. [12] and Peeters et al. [13], respectively, proposed collaborative authentication protocols based on threshold cryptography, where multiple user devices jointly perform user authentication to remote verifiers. These approaches commonly use a secret sharing technique to split a secret key for user authentication into multiple shares and distribute them over multiple user devices. Due to the nature of the secret sharing technique in these approaches, the security of the collaborative user authentication process can be improved as the number of user devices sharing the secret key increases. However, the maintenance cost of secret shares over multiple user devices also increases because the maintenance operation requires communication among the user devices; according to [12], the maintenance cost of sharing a secret takes about t^2 to $(n-1)^2$ under the assumption of (t, n)-threshold scheme, where at least t out of n user devices are required for user authentication. In this paper, we considered a more realistic and practical deployment achieving a balance between the security and the convenience. In our system, we particularly design a secret sharing scheme using PBKDF (Password-based key derivation function). Specifically, our approach encrypts a user's sign key and physically separates the secret parameters used to generate the encryption key into the primary and the secondary user devices. With this simplified system model, the proposed system can reduce the management cost while achieving a reasonable level of security. In addition, unlike

previous studies, we implemented a fully working system and evaluated its performance in real network settings.

Certificate-based authentication has been still widely used in a variety of applications. Hiltgen et al. [14] presented a certificate-based user authentication method of leveraging smart cards to mitigate the threats on user authentication in Internet banking applications. Certificate-based authentication is also commonly used by open source IoT frameworks (e.g., AllJoyn [15], OCF (Open Connectivity Foundation) specification [16], IoTivity [17], etc.) to establish secure sessions between devices while minimizing user intervention. To the best of our knowledge, no previous study has presented an offline guessing attack scheme against certificate-based authentication with the goal of obtaining a user's private key from the password-protected private key file. As such, we analyzed this type of threat in the context of user authentication to industrial controllers, and developed a countermeasure to mitigate the threat.

3. Certificate-Based Authentication

In this section, we explain how a digital certificate can be used for user authentication in an industrial controller application. A conventional certificate-based user authentication scheme [18] can be described at a high level as follows.

Suppose that a user u uses a device d (e.g., smartphone or laptop) with a hardware token holding the digital certificate $cert_u$, and its corresponding sign key sk_u for an industrial controller c. To protect sk_u in d, d stores sk_u in an encrypted form (i.e., $ENC(k, sk_u)$). An encryption key k is generated by using a key-derivation function $KDF(\cdot)$ with the user's password $pass_u$ (i.e., $k = KDF(pass_u)$). Figure 1 shows the basic user authentication procedure.

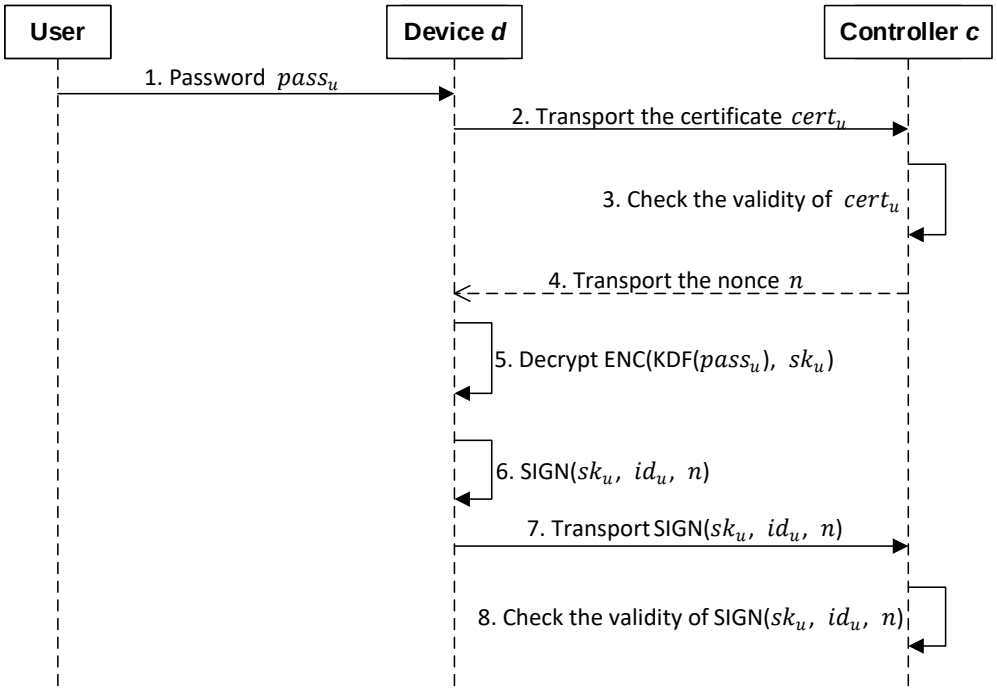

Figure 1. User authentication using a digital certificate.

To perform the authentication, the user u should enter his/her password through the application installed on the device d (see Step 1 in Figure 1). Next, d sends a login request message containing the user's digital certificate $cert_u$ (again containing the user identity id_u and its public (verification)

key pk_u) to the controller c (see Step 2 in Figure 1). After receiving the login request message, c extracts $cert_u$ from the message and verifies its validity by checking whether it was issued by a trusted certificate authority and has not expired or been revoked. If the received $cert_u$ is valid, c replies to the request message by returning a random nonce n (see Steps 3 and 4 in Figure 1). After receiving n, d computes the encryption key k; if the entered password in Step 1 in Figure 1 is the same as $pass_u$, k is successfully generated by computing $KDF(\cdot)$ with the entered password, and k is then used to decrypt $ENC(k, sk_u)$. Therefore, sk_u is loaded into the memory in d during runtime (see Step 5 in Figure 1). Next, d digitally signs id_u and n with the user's sign key sk_u (i.e., $SIGN(sk_u, id_u, n)$) and replies to c (see Step 6 and 7 in Figure 1). After receiving $SIGN(sk_u, id_u, n)$ from d, c verifies its validity with the user's public key pk_u. If the received message is valid, the user u can login to c for a transaction (see Step 8 in Figure 1).

4. Threat Model

We consider an attacker who can access the data stored in the user device d, for example, by stealing d, and the attacker can obtain the encrypted sk_u from d. In addition, the attacker can obtain the nonce n from controller c and the corresponding signature generated by device d by monitoring the traffic between d and c. However, we assume that the attacker does not know the user's password $pass_u$. The attacker is also assumed to be computationally bounded to polynomial time, which in particular prevents the attacker from breaking computationally-secure cryptographic primitives (e.g., AES [19]). However, the attacker fully understands the description of all procedures for user authentication and key protection. In practice, such assumptions are often valid (e.g., [20]).

The attacker's goal is to obtain the user's sign key sk_u. Since sk_u is encrypted with key k, which is generated with the user's password $pass_u$ alone, the security of sk_u is only as good as the user's password, $pass_u$, which is inherently susceptible to *offline* password guessing attacks. That is, if the attacker holds any password-related data, they can then try to iteratively guess the user's password and check whether the guess is correct by trying to calculate the password-related data from the guess. Even though password complexity policies (e.g., requirements for minimum length, symbols, numbers, and upper/lowercase letters) or password strength meters are used to encourage users to choose strong passwords, advanced attackers are still developing methods to efficiently crack such passwords offline using various forms of hybrid password guessing attacks [21]. The basic attack procedure is as follows:

1. An attacker captures a nonce n and signed message $SIGN(sk_u, id_u, n)$ by observing the traffic between the device d and controller c.
2. The attacker also accesses d and obtains the user's certificate $cert_u$ and encrypted user's sign key $ENC(KDF(pass_u), sk_u)$; the attacker then extracts the user identity id_u from $cert_u$.
3. With a candidate password $\hat{p}$, the attacker calculates $\hat{k} = KDF(\hat{p})$ and then tries to decrypt $ENC(KDF(pass_u), sk_u)$ with $\hat{k}$. We use the notation $\hat{sk}$ to represent the decrypted result. Finally, the attacker checks whether $SIGN(\hat{sk}, id_u, n)$ is the same as $SIGN(sk_u, id_u, n)$, which means that the candidate password $\hat{p}$ is correctly guessed. This procedure is repeated with a new candidate password $\hat{p}$ until $SIGN(\hat{sk}, id_u, n)$ is the same as $SIGN(sk_u, id_u, n)$.

5. Proposed User Authentication System

Under the threat model described in Section 4, we showed that the existing certificate-based authentication system could be vulnerable to *offline* password guessing attacks. This is because the stored key data are encrypted only with the user's password in such systems.

We suggest a new method to protect the user's sign key, sk_u, from offline guessing attacks. The encryption key can only be generated with the assistance of a secondary device. To develop a fully working system, we considered two types of secondary devices: (1) an Internet server and (2) another user device that is nearby via a short-range network channel (e.g., Bluetooth or ZigBee).

We assume that the user device d is connected to the Internet in most cases through Wi-Fi or the mobile device's LTE/4G connection. In some situations, however, d cannot be connected directly to

the Internet. First, even though LTE/4G connection can cover most places in which controllers are located, the cost of LTE/4G data service plans is not widely in favor of end users; hence, they may resist using the LTE/4G service. Second, many client devices such as laptops do not directly support LTE/4G radios and hence rely on Bluetooth or Wi-Fi to send/receive data. Third, the service coverage is not guaranteed in certain areas (e.g., rural or underground buildings). Therefore, we considered a secondary device equipped with short-range communication as a backup plan in case an Internet server is unreachable from device d, for example, while it is being restored.

The proposed user authentication system consists of five components: user u, device d, controller c, salt server s, and mobile phone m. Device d could be a mobile phone or laptop of user u and refers to a separate device from m. Salt server s and mobile phone m in the proposed system provide two complementary options to obtain the additional parameters, i.e., $salt^1$ and $salt^2$, which are required to generate the encryption keys, i.e., k^1 and k^2, respectively. The only difference between s and m is that s is connected with d via the Internet and m is connected with d via short-range communication. It is assumed that d has secure network sessions with s and m via, for example, SSL/TLS and Bluetooth protocols. The proposed system also has two phases: (1) encryption key generation and (2) user authentication. We present the detailed procedure of each phase in the following sections.

5.1. Encryption Key Generation

We assume that the user u initially holds his/her digital certificate $cert_u$ and its corresponding sign key sk_u for an industrial controller c. To securely protect sk_u, we encrypt sk_u, which is the same as conventional certificate-based authentication systems (see Section 3). However, unlike conventional systems, we introduce an additional secret parameter called *random salt* that is randomly chosen. The parameter $salt$ is used together with $pass_u$ to generate the encryption key $k = KDF(pass_u, salt)$. To securely protect k against offline password guessing attacks, $salt$ is stored in another device (e.g., semi-trusted server or smartphone) that the user u can connect to. The steps of this procedure are illustrated in Figure 2 and are explained as follows:

1. The user u enters his or her password, $pass_u$, through the application installed on the device d.
2. To register the encryption key, d needs to connect to the salt server s, mobile device m, and controller c. If any of these are not connected, the encryption key registration process fails.
3. After d connects to all of the servers, an additional two random salts, $salt^1$ and $salt^2$, are generated.
4. u stores $salt^1$ and $salt^2$ in separate places (e.g., the salt server s and mobile device m) rather than their own device d to mitigate offline guessing attacks against the user's sign key sk_u. For this, d sends the user account registration message to s. The user account registration message contains the user identity id_u, password $pass_u$, and system generated (random) parameter $salt^1$ (see Step 4a—1 in Figure 2). We assume here that s is available and can be connected from device d. After receiving the user account registration message, s stores the triple (id_u, $HASH(pass_u)$, $salt^1$) into its user database. In a similar manner, the triple (id_u, $HASH(pass_u)$, $salt^2$) is registered to the user's other mobile device m under the assumption that m is available and can be connected from device d via short range communication technology (e.g., Bluetooth or ZigBee). d sends the certificate $cert_u$ to controller c (see Step 4c—1 in Figure 2). c then stores the received certificate $cert_u$.
5. Using $pass_u$ along with the random salts $salt^1$ and $salt^2$, the device d generates two encryption keys k^1 and k^2 by calculating $k^1 = KDF(pass_u, salt^1)$ and $k^2 = KDF(pass_u, salt^2)$, respectively (see Steps 5a and 5b in Figure 2).
6. After generating k^1 and k^2, sk_u is encrypted with them, i.e., $ENC(k^1, sk_u)$ and $ENC(k^2, sk_u)$, respectively, and stored in the device d.

If d fails to connect to both s and m, the proposed system can rollback the committed transactions for the key generation procedure.

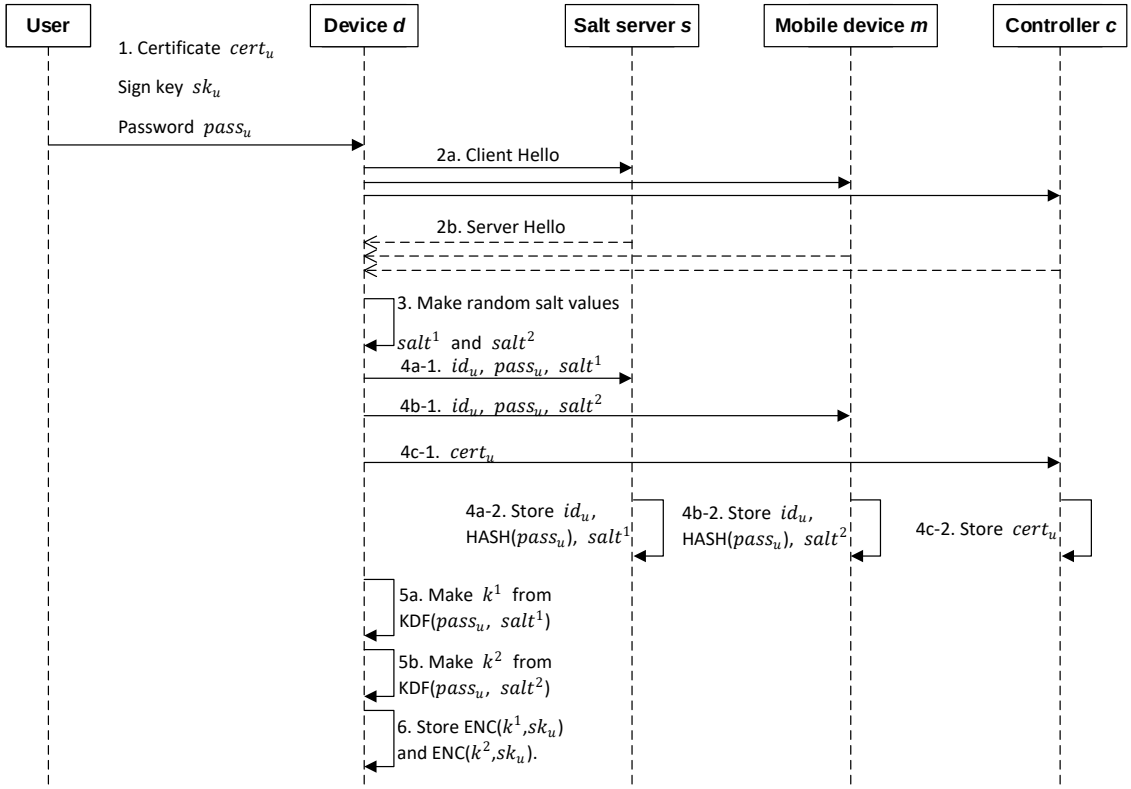

Figure 2. Encryption key generation procedure.

5.2. User Authentication

After finishing the steps described in Section 5.1, the user authentication procedure can be performed for the user u to obtain access to the controller c. The user authentication procedure is illustrated in Figure 3. An explanation of each step of this procedure is as follows:

1. For the user authentication, the user u should first enter his/her password $pass_u$ through the application installed on the device d (see Step 1 in Figure 3).
2. To decrypt the encrypted user's sign key sk_u, d has to obtain either one of the salts ($salt^1$ or $salt^2$) from the salt server s or mobile device m. Therefore, to obtain a salt, d first tries to connect to s (See Figure 3). If s does not respond to d, d sequentially tries to connect to m. If any of these are not connected, the authentication process fails because d cannot decrypt sk_u.
3. If d is successfully connected with s, d sends the salt request message containing the tuple (id_u, $pass_u$) to s (see Step 3a in Figure 3). After receiving the salt request message, s retrieves the user record matching with id_u from its user database and checks the validity of $pass_u$ by comparing its hash value with the stored password hash value in the matching record. To mitigate password guessing attacks, we recommend the use of a security policy to limit the number of failed password attempts and/or impose a time interval between failed attempts. After a maximum number of consecutive salt request attempts with invalid passwords, further attempts are blocked completely. Thus, this feature ensures that offline password guessing attacks are infeasible, unlike existing certificate-based authentication methods.
4. If the received $pass_u$ is valid, s replies to the salt request message by returning the stored salt $salt^1$ (see Step 4 in Figure 3). Here, the communication channel between d and s is assumed to be a secure and authenticated channel (e.g., SSL/TLS). If s is unavailable, the user's mobile device m can be alternatively used instead of s. If the received $pass_u$ is valid, m replies to d with the stored

salt $salt^2$. Without loss of generality, we assume here that s is available because further processes are the same in both cases.

5. After receiving $salt^1$ (or $salt^2$), d computes $k^1 = KDF(pass_u, salt^1)$ (or $k^2 = KDF(pass_u, salt^2)$) to extract the encryption key k^1 (or k^2) from $ENC(k^1, sk_u)$ (or $ENC(k^2, sk_u)$). Again, d decrypts $ENC(k^1, sk_u)$ with k^1 (or $ENC(k^2, sk_u)$ with k^2) to obtain the user's sign key sk_u for user authentication.
6. The device d sends a login request message containing the user u's digital certificate $cert_u$ (again containing user identity id_u and its public key pk_u) to the controller c (see Step 6a in Figure 3). After receiving the login request message, c extracts $cert_u$ from the message and verifies its validity by checking whether it was issued by a trusted certificate authority and has not expired or been revoked. If the received $cert_u$ is valid, c replies to the request message by returning a random nonce n (see Steps 6b and 6c in Figure 3).
7. After receiving the nonce n, d digitally signs the user identity id_u and n with the user's sign key sk_u (i.e., $SIGN(sk_u, id_u, n)$) and replies to c (see Steps 7a and 7b in Figure 3). After receiving $SIGN(sk_u, id_u, n)$ from d, c verifies its validity with the user's public key pk_u. If the received signed message is valid, the user (u) can login to c for a transaction.

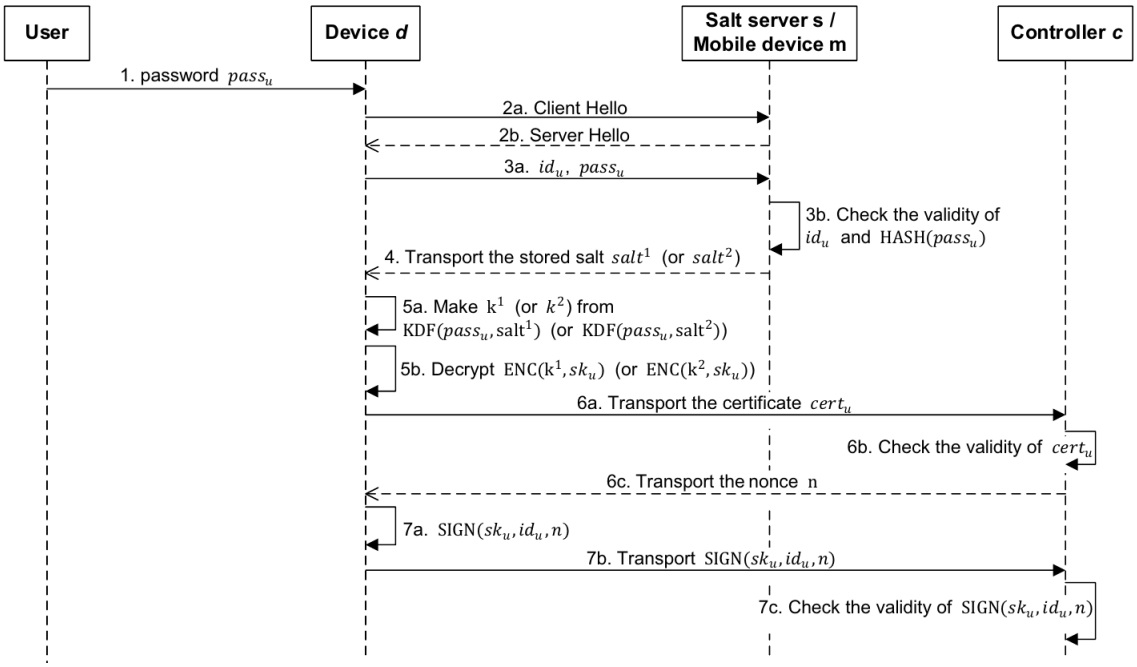

Figure 3. User authentication procedure using either the salt server (with $salt^1$) or the mobile device (with $salt^2$) in the proposed system.

6. Security Evaluation with ProVerif

We present a brief formal security evaluation of the proposed protocols described in Section 5 by ProVerif [22] that is popularly used for verifying cryptographic protocols (e.g., [23,24]). Under an active Dolev–Yao adversary [25], ProVerif can verify the confidentiality and authenticity of the compromised participant, Mallory, that is also allowed to initialize sessions and exchange messages with network entities.

To simplify our model in ProVerif, we use a private channel to represent secure sessions based on SSL/TLS and BLE.

6.1. Key Enrollment Procedure

We modeled a set of cryptographic primitives F_c that are used in our proposed key protection system (see the ProVerif code in Listing 1).

Listing 1. List of functions required for the key enrollment process.

```
Functions:
F_c={HASH, KDF, ENC}
HASH(data) = hashed value.
KDF(data, salt, HASH, iteration) = hashed value.
ENC(Encryption key, data) = ciphertext.
```

In the salt server s or mobile device m, a random salt value (i.e., either $salt^1$ or $salt^2$) is stored along with the user's ID and password. The corresponding ProVerif expression is provided in Listing 2.

Listing 2. Procedure performed on the salt server s or mobile device m for the key enrollment.

```
Store (id_u, HASH(pass_u), salt^1 or salt^2)
```

In the device d, the user's sign key sk_u is stored after being encrypted with a key-value derived from the user's password and random salt value. The corresponding ProVerif expression is provided in Listing 3.

Listing 3. Procedure performed on device d for the key enrollment.

```
k^1 = KDF(pass_u, salt^1, HASH, 10000)
k^2 = KDF(pass_u, salt^2, HASH, 10000)
Store ENC(k^1, sk_u), ENC(k^2, sk_u), cert_u
```

For these settings, the proposed system can successfully defeat an attacker who is unable to compromise both the user device d and salt server s (or the mobile device m), simultaneously. For example, if only d is stolen, the attacker can only obtain the ciphertext of the user's sign key sk_u, but neither the user's password $pass_u$ nor a salt value $salt^1$ (or $salt^2$). Therefore, it is infeasible for the attacker to obtain sk_u from the ciphertext. Similarly, if only s or m are stolen, the attacker can only obtain the user id id_u, the hash value of the password HASH($pass_u$), and a salt value $salt^1$ (or $salt^2$). This knowledge is also insufficient for obtaining sk_u from the ciphertext.

6.2. User Authentication

When the user tries to access the controller c, the user's sign key sk_u is needed. We modeled a set of cryptographic primitives F_c and F_d that are used in our key protection system, and are defined in Listing 4.

The user's sign key sk_u is encrypted with the key derived from the user's password and random salt value, and the encrypted key is then stored in d. The user only knows their ID and password, but not the salt value. Therefore, d should request the salt value for the salt server s if the device is connected to the Internet or mobile device m.

After receiving the salt value from s or m, d computes the decryption key by performing the key-derivation function (KDF) for the user's password $pass_u$ and salt and decrypts the ciphertext of sk_u with the decryption key.

To obtain the access permission to the controller c, d receives the nonce n from c, signs the user's ID and n with the decrypted user's sign key sk_u, and then sends the signed value to c. If the received signature is verified successfully, c responds d with the symmetric key which will be used to secure subsequent communication between them. A brief representation in ProVerif is shown in Listing 5.

Listing 4. List of functions required for the user authentication process.

```
Functions:
F_c={GetPKey, ENC, SIGN, GenCert, KDF, HASH}
HASH(data) = hashed value.
KDF(data, salt, HASH, iteration) = hashed value.
ENC(Encryption key, data) = cipher text.
GetPKey(Private key) = Public key.
GenCert(Public key, Private key) = Certificate.
SIGN(Private key, data) = signed value.
F_d={DEC, CheckSIGN, ExtractPKey}
DEC(Key, cipher text) = data.
CheckSIGN(Signed value, Public key) = True or False.
ExtractPKey(Certificate) = Public key.
```

Listing 5. Procedure performed on device d for the user authentication.

```
Stored information:
ENC(k^1, sk_u)
ENC(k^2, sk_u)
Certificate cert_u
Process: Device≙
out(c_priv, (id_u, pass_u)).
in(c_priv, salt^1 or salt^2).
let k̂ = KDF(pass_u, salt, HASH, 10000) in
let dec_sk_u = DEC(k̂, ENC(k^1 or k^2, sk_u)) in
out(c, Certificate).
in(c, n).
out(c, SIGN(dec_sk_u, (id_u, n))).
in(c, encrypted value).
let key = DEC(dec_sk_u, encrypted value) in
out(c, ENC(key, m)).
```

In the salt server s or mobile device m, the user credential information and salt value were registered in the key enrollment procedure. These are ready to receive the user ID id_u and password $pass_u$ from d. If s or m find a match with the received information from d, they respond to d with the stored salt value $salt^1$ (or $salt^2$). The ProVerif expression is shown in Listing 6.

Listing 6. Procedure performed on the salt server s or mobile device m for the user authentication.

```
Stored information:
(id_u, HASH(pass_u), salt^1 or salt^2)
Process: Salt≙
in(c_priv, (îd_u, pâss_u)).
if îd_u = id_u then
if HASH(pâss_u) = HASH(pass_u) then
out(c_priv, salt^1 or salt^2).
```

In the user authentication process, c receives a certificate from d and checks the validity of the certificate. If the received certificate is valid, c sends a random nonce n to d as a challenge and waits for a response from d. Once receiving the signature of n from d, c verifies the signature with the

user's public key. If the signature is valid, c sends the symmetric key to d for securing subsequent communication. The brief ProVerif expression is provided in Listing 7.

Listing 7. Procedure performed on controller c for the user authentication.

```
Stored information:
Certificates
Process: Controller≙
in(c, Certificate).
New n.
out(c, n).
let Public key = ExractPKey(Certificate) in
in(c, signed value).
if CheckSIGN(signed value) then
New key.
out(c, ENC(Public key, key)).
```

6.3. ProVerif Results

To verify the security and correctness of our proposed protocols, we ran ProVerif on the codes described in Sections 6.1 and 6.2. We set the attacker as an active Dolev–Yao adversary [25]. In addition, to verify that our protocols are safe under the threat model described in Section 4, we assumed four cases: standard, information leakage from d, information leakage from m, and information leakage from both d and m.

Standard case. As we mentioned at the beginning of Section 6, we used the private channel when communicating between the mobile and user devices, and used the public channel when communicating between the user's device and controller. As a result, ProVerif showed that the attacker cannot obtain the user's sign key sk_u.

Information leakage from the user's device d alone. In this case, we assumed that an attacker could access the data stored in d such as id_u, encrypted user's sign key $\texttt{ENC}(k, sk_u)$, and $cert_u$. To mimic this data leakage, we sent the data over the public channel. According to ProVerif results, the attacker cannot still obtain the user's sign key sk_u.

Information leakage from the mobile device, m, alone. In this case, we assumed that an attacker could access the data stored in m such as id_u, $\texttt{HASH}(pass_u)$, and $salt^1$ or $salt^2$. To mimic this data leakage, we sent the data over the public channel. The ProVerif results show that the attacker still cannot obtain the user's sign key sk_u.

Information leakage from both d and m. In this case, we assumed that an attacker can access the data stored in both d and m, including id_u, $\texttt{ENC}(k, sk_u)$, $\texttt{HASH}(pass_u)$, $salt^1$ or $salt^2$, and $cert_u$. To mimic this case of data leakage, we sent the data over the public channel. The results of ProVerif show that the attacker can successfully obtain the user's sign key sk_u. The detailed process for obtaining sk_u is shown in Listing 8.

Listing 8. Attack trace for information leakage from both the user and mobile devices, d and m, respectively.

```
d → ATTACKER : ENC(k, sk_u)
m → ATTACKER : HASH(pass_u), salt
By dictionary attack, ATTACKER get pass_u from HASH(pass_u)
ATTACKER : generate k using KDF function with pass_u and salt
ATTACKER : With k, get sk_u from ENC(k, sk_u) using DEC function
```

Even without active intervention, the attacker can obtain the user's sign key sk_u, but only if the attacker can access the data stored in both d and m to obtain $\texttt{ENC}(k, sk_u)$, $\texttt{HASH}(pass_u)$, and $salt$ together. The attacker can use the dictionary attack to successfully guess $pass_u$ from $\texttt{HASH}(pass_u)$. Using $pass_u$ and $salt$, the attacker can then make the encrypted key k using the key-derivation function $\texttt{KDF}(pass_u, salt, \texttt{HASH}, 10{,}000)$. Using sk_u, the attacker can mimic the user with the user's digital certificate and freely access the controller c.

7. Implementation and Evaluation

To show the feasibility and effectiveness of the proposed user authentication system, we implemented a prototype and performed several experiments with the prototype implementation. The implementation and performance evaluation of the proposed system is described in this section. As a baseline for the comparison, we also implemented the conventional system (described in Section 3) and performed the same experiments.

7.1. Implementation

For digital signature algorithms, we used RSA and ECDSA with SHA-256 and respectively used 3072 bits and 256 bits for the length of a public key pk_u which achieve the same level of security according to the recommendation of NIST [26]. To securely store the user's sign key sk_u, sk_u was encrypted using the AES-256 algorithm in Galois/Counter Mode (GCM) without padding. We note that padding is not required because random keys are encrypted. To derive the AES encryption key, PBKDF2 with HMAC-SHA256 was used where the number of iterations was 10,000, and the salt was generated by a cryptographically secure pseudorandom number generator (CSPRNG). Similarly, CSPRNG was also used to generate the initial vector for the AES encryption in GCM. We used a SHA3-256 hash function to compute the hash value of the user password.

As shown in Figure 2, the proposed user authentication system consists of three communication channels between the user device and salt server, '$d - s$', user device and mobile device, '$d - m$'; and user device and controller, '$d - c$'. For the salt server implementation, we used Spring Boot (https://spring.io/projects/spring-boot) to build the Representational State Transfer (REST) architecture. Therefore, HTTPS was used for the channel '$d - s$'. For the channels '$d - m$' and '$d - c$', short-range communication based on Bluetooth with Serial Port Profile (SPP) was used.

For further information, the source codes of our implementation are available in the public GitHub repository (https://github.com/rymuff/keyprotection, see Supplementary Materials).

7.2. Times Taken for Key Enrollment and Authentication

As described in Section 5, our user authentication system consists of four components; user device d, salt server s, mobile device m, and controller c. In the following experiments, we used a MacBook Pro with Intel Core i7-7920HQ (3.10 GHz) running on macOS Catalina for the user device, a desktop with AMD Ryzen Threadripper 1950X (3.40 GHz) running on Windows 10 Pro for the controller, a desktop with Intel Core i7-9700 (3.00 GHz) running on Ubuntu 18.04 for the salt server, and a Google Pixel 3a running on Android 9 (Pie) as the mobile device. The evaluation results are presented in the following sections.

7.2.1. Key Enrollment Overhead

To evaluate the latency caused by the key enrollment procedure in our system, we measured the execution time of the entire procedure described in Figure 2 including the key generation time. We also performed the same measurements on the conventional user authentication system described in Section 3 to compare the overheads. For the key enrollment procedure in the conventional system, the user's sign key was generated and simply encrypted with the user's password. For both systems, we repeated the same experiments 100 times. Table 1 summarizes the results.

Table 1. Times (seconds) for key enrollment and authentication (C: Conventional System, P: Proposed System, S: Salt Server, M: Mobile Device).

Procedure	Key Enrollment				Authentication					
Method	C		P		C		P with S		P with M	
Algorithm	RSA	ECDSA	RSA	ECDSA	RSA	ECDSA	RSA	ECDSA	RSA	ECDSA
Avg.	0.326	0.014	0.658	0.279	0.383	0.319	0.422	0.392	0.634	0.437
Std.	0.209	0.002	0.295	0.038	0.140	0.030	0.051	0.038	0.344	0.222
Min.	0.086	0.011	0.324	0.235	0.294	0.270	0.355	0.331	0.303	0.266
Max.	1.133	0.025	1.849	0.469	1.646	0.400	0.688	0.530	1.624	1.278
Med.	0.282	0.013	0.597	0.262	0.359	0.320	0.416	0.393	0.506	0.344

For the proposed system, the average execution times were 0.658 s for RSA-3072 and 0.279 s for ECDSA-256 with respective standard deviations of 0.295 s and 0.038 s. For the conventional system, on the other hand, the average execution times were 0.326 s for RSA-3072 and 0.014 s for ECDSA-256 with respective standard deviations of 0.209 s and 0.002 s. Even though the execution times of our system were about 2 and 20 times longer than the conventional system, the absolute values of the performance gap were as small as 0.333 s and 0.266 s on average. Furthermore, the enrollment procedure was performed only once and can be amortized over the subsequent authentication procedures.

To find a more suitable digital signature algorithm for our system, we implemented both RSA-3072 and ECDSA-256 and compared the execution times of both implementations. As shown in Table 1, the execution time for the key enrollment procedure of the implementation using ECDSA-256 was 0.379 s shorter than the implementation using RSA-3072.

7.2.2. Authentication Overhead

We also evaluated the performance of the user authentication procedure in the proposed system. For the user authentication, the user device, d, tries to obtain the salt value either from the salt server, s, or mobile device, m, depending on the connectivity to the salt server.

We first assumed that the user device has Internet connectivity so that it can be connected to the salt server. Under this assumption, we measured the execution time of the authentication procedure using the salt server described in Figure 3. The 'P with S' column of *Authentication* in Table 1 summarizes the results of 100 measurements. The average execution times of the proposed system were 0.422 s and 0.392 s with a standard deviation of 0.051 s and 0.038 s, respectively, for RSA-3072 and ECDSA-256. As shown in Table 1, the execution time of the user authentication procedure in the proposed method was slower than that of the user authentication procedure in the conventional system, without requiring any extra communication to obtain a salt (see Figure 1). However, the execution time overhead incurred by the user authentication procedure for the proposed system was comparable to that of the conventional system, indicating that most casual users cannot recognize such a time difference.

Next, we assumed that the user device, d, has no connectivity to the salt server. In this case, the user device provides the user with a notification that the connection to the salt server has failed, and then tries to retrieve the salt value $salt^2$ from the mobile device, m, via the Bluetooth connection. The authentication procedure using the mobile device instead of the salt server in Figure 3 corresponds to this case. In this case, d performs the steps from 2a to 5b with the mobile device in Figure 3. Again, we measured the execution time of this procedure 100 times. The 'P with M' column of *Authentication* in Table 1 summarizes the measurement results. The execution times of the proposed system were 0.634 s for RSA-3072 and 0.437 s for ECDSA-256 on average, which are slower than the execution times of the conventional system. In fact, this extra overhead mainly came from the Bluetooth connection setup time. In addition, this overhead is incurred only when the user device had no connectivity to the salt server, and we claim that this is a rare case in practice.

Again, for the user authentication procedure, we compared the execution times of both implementations using RSA-3072 and ECDSA-256. For the proposed system, the ECDSA-256 based

implementation (0.392 s for the salt server case and 0.437 s for the mobile device case) showed a slightly better performance than the RSA-3072 based implementation (0.422 s for the salt server case and 0.634 s for the mobile device case). According to these evaluation results, ECDSA-256 would be a more preferable recommendation for our system.

8. Conclusions

We present a formally verified certificate-based user authentication method using a secondary network device to protect a user's sign key. In the proposed method, the user's sign key is encrypted and can only be decrypted with the assistance of a secondary device. To show the feasibility of the proposed method, we implemented a prototype with popularly used cryptographic algorithms (AES-256, RSA-3072, and ECDSA-256). Through the security evaluation using ProVerif and the performance evaluation of the prototype implementation, we confirmed that the proposed method can bring significant benefits with respect to security with only a slight sacrifice in performance (0.073 s increase with respect to the execution time for the user authentication procedure).

As part of future work, we plan to conduct real-world experiments through the deployment of the proposed method with a company that is developing industrial controller solutions, and analyze its usability issues (e.g., user satisfaction and network stability) in real-world settings.

Supplementary Materials: The source code of our implementation and the data files of the evaluation results are available at a GitHub repository (https://github.com/rymuff/keyprotection). The GitHub repository contains the URL links to the source codes of the client application, the salt server, the mobile application, and the controller. The repository also contains the URL links to the following data files of the evaluation results: the data files of the analysis results of the authentication using a conventional system and the proposed system and the data files of the execution time and power consumption measurements of the backdoor application.

Author Contributions: Methodology, J.C. (Jusop Chio) and S.H.; Software, J.C. (Junsung Cho); Formal Analysis, J.C. (Jusop Chio); Data Curation, J.C. (Junsung Cho); Writing—Original Draft Preparation, J.Choi and H.K.; Writing—Review & Editing, S.H.; Supervision, H.K. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the MIST (Ministry of Science and ICT), Korea, under the ICT Consilience Creative program (IITP-2019-2015-0-00742) supervised by the IITP (Institute for Information & communications Technology Planning & Evaluation) and by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. 2019R1C1C1007118).

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AES	Advanced Encryption Standard
IIoT	Industrial Internet of Things
PBKDF2	Password-Based Key Derivation Function 2
SSL	Secure Socket Layer
TLS	Transport Layer Security
LTE/4G	Long-Term Evolution
ECDSA	Elliptic Curve Digital Signature Algorithm
NIST	National Institute of Standards and Technology
GCM	Galois/Counter Mode
HMAC	Hash-based Message Authentication Code
CSPRNG	Cryptographically Secure Pseudorandom Number Generator
SHA3	Secure Hash Algorithm
REST	Representational State Transfer
SPP	Serial Port Profile
ICS	Industrial Control System
QR code	Quick Response code
sk_u	Sign key of user

id_u	Identification number of user
$pass_u$	Password of user
n	Nonce
$salt$	Salt value
k	Symmetric key
$cert_u$	Certificate of user
ENC	Encryption
KDF	Key Derivation Function
$SIGN$	Signing
$HASH$	Hash algorithm
$GetPKey$	Generate public key from private key
$GenCert$	Generate certificate
DEC	Decryption
$CheckSIGN$	Check the signed data with public key
$ExtractPKey$	Extract public key from certificate

References

1. Huh, J.H.; Bobba, R.B.; Markham, T.; Nicol, D.M.; Hull, J.; Chernoguzov, A.; Khurana, H.; Staggs, K.; Huang, J. Next,-generation access control for distributed control systems. *IEEE Internet Comput.* **2016**, *20*, 28–37. [CrossRef]
2. Stajano, F. Pico: No more passwords! In Proceedings of the International Workshop on Security Protocols, Cambridge, UK, 28–30 March 2011.
3. Kaliski, B. *PKCS# 5: Password-Based Cryptography Specification Version 2.0*; RFC 2898; RSA Laboratories: Bedford, MA, USA, 2000.
4. Canetti, R.; Halevi, S.; Steiner, M. Mitigating Dictionary Attacks on Password-Protected Local Storage. In Proceedings of the 26th Annual International Cryptology Conference, Santa Barbara, CA, USA, 20–24 August 2006.
5. Catuogno, L.; Galdi, C.; Riccio, D. Off-line enterprise rights management leveraging biometric key binding and secure hardware. *J. Ambient Intell. Humaniz. Comput.* **2019**, *10*, 2883–2894. [CrossRef]
6. Catuogno, L.; Galdi, C. A Fine-grained General Purpose Secure Storage Facility for Trusted Execution Environment. In Proceedings of the International Conference on Information Systems Security and Privacy, Prague, Czech Republic, 23–25 February 2019.
7. Tiago, A.; Don, F. TrustZone: Integrated hardware and software security enabling trusted computing in embedded system. *Gov. Inf. Q.* **2004**, *3*, 18–24.
8. Stouffer, K.; Falco, J.; Scarfone, K. *NIST Special Publication 800-82: Guide to Industrial Control Systems (ICS) Security*; NIST: Gaithersburg, MD, USA, 2011.
9. Borisov, A. A Novel Approach for User Authentication to Industrial Components Using QR Codes. In Proceedings of the 2015 IEEE 39th Annual Computer Software and Applications Conference, Taichung, Taiwan, 1–5 July 2015.
10. Plaga, S.; Niethammer, M.; Wiedermann, N.; Borisov, A. Adding Channel Binding for an Out-of-Band OTP Authentication Protocol in an Industrial Use-Case. In Proceedings of the 2018 1st International Conference on Data Intelligence and Security (ICDIS), South Padre Island, TX, USA, 8–10 April 2018.
11. Bhargavan, K.; Delignat-Lavaud, A.; Pironti, A.; Langley, A.; Ray, M. Transport Layer Security (TLS) Session Hash and Extended Master Secret Extension. 2015. Available online: https://rfc-editor.org/rfc/rfc7627.txt (accessed on 30 June 2019).
12. Abidin, A.; Aly, A.; Mustafa, M.A. Collaborative Authentication Using Threshold Cryptography. In *International Workshop on Emerging Technologies for Authorization and Authentication*; Springer: Cham, Switzerland, 2019; pp. 122–137.
13. Peeters, R.; Singelee, D.; Preneel, B. Toward more secure and reliable access control. *IEEE Pervasive Comput.* **2011**, *11*, 76–83. [CrossRef]
14. Hiltgen, A.; Kramp, T.; Weigold, T. Secure Internet Banking Authentication. *IEEE Secur. Priv.* **2006**, *4*, 21–29. [CrossRef]
15. AllJoyn Framework. Available online: https://certify.alljoyn.org/ (accessed on 30 June 2019).

16. The OCF Security Specification. 2017. Available online: https://openconnectivity.org/specs/OCFSecuritySpecificationv1.0.0.pdf (accessed on 30 June 2019).
17. IoTivity Wiki. Available online: https://wiki.iotivity.org (accessed on 30 June 2019).
18. Weigold, T.; Kramp, T.; Baentsch, M. Remote Client Authentication. *IEEE Secur. Priv.* **2008**, *6*, 36–43. [CrossRef]
19. Daemen, J.; Rijmen, V. *The Design of Rijndael*; Springer: New York, NY, USA, 2002.
20. Choi, J.; Park, J.; Kim, H. Forensic analysis of the backup database file in KakaoTalk messenger. In Proceedings of the IEEE International Conference on Big Data and Smart Computing, Jeju, Korea, 13–16 February 2017.
21. Huh, J.H.; Oh, S.; Kim, H.; Beznosov, K.; Mohan, A.; Rajagopalan, S.R. Surpass: System-initiated user-replaceable passwords. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, 12–16 October 2015.
22. Blanchet, B. Automatic verification of correspondences for security protocols. *J. Comput. Secur.* **2009**, *17*, 363–434. [CrossRef]
23. Chaudhry, S.A.; Farash, M.S.; Naqvi, H.; Sher, M. A secure and efficient authenticated encryption for electronic payment systems using elliptic curve cryptography. *Electron. Commer. Res.* **2016**, *16*, 113–139. [CrossRef]
24. Cortier, V.; Galindo, D.; Turuani, M. A Formal Analysis of the Neuchatel e-Voting Protocol. In Proceedings of the 2018 IEEE European Symposium on Security and Privacy, London, UK, 24–26 April 2018
25. Dolev, D.; Yao, A. On the security of public key protocols. *IEEE Trans. Inf. Theory* **1983**, *29*, 198–208. [CrossRef]
26. Barker, E. Recommendation for Key Management. 2019. Available online: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5-draft.pdf (accessed on 22 October 2019).

© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (http://creativecommons.org/licenses/by/4.0/).