Leakage-Resilient and Proactive Authenticated Key Exchange (LRP-AKE), Reconsidered

SUMMARY  In [31], Shin et al. proposed a Leakage-Resilient and Proactive Authenticated Key Exchange (LRP-AKE) protocol for credential services which provides not only a higher level of security against leakage of stored secrets but also secrecy of private key with respect to the involving server. In this paper, we discuss a problem in the security proof of the LRP-AKE protocol, and then propose a modified LRP-AKE protocol that has a simple and effective measure to the problem. Also, we formally prove its AKE security and mutual authentication for the entire modified LRP-AKE protocol. In addition, we describe several extensions of the (modified) LRP-AKE protocol including 1) synchronization issue between the client and server’s stored secrets; 2) randomized ID for the provision of client’s privacy; and 3) a solution to preventing server compromise-impersonation attacks. Finally, we evaluate the performance overhead of the LRP-AKE protocol and show its test vectors. From the performance evaluation, we can confirm that the LRP-AKE protocol has almost the same efficiency as the (plain) Diffie-Hellman protocol that does not provide authentication at all.

key words: password, two-factor authentication, key exchange, leakage of stored secrets, credential services, provable security

1. Introduction

An Authenticated Key Exchange (AKE) protocol is a fundamental cryptographic primitive by which involving parties can authenticate each other and generate secure session keys for protecting subsequent messages to be exchanged. For several decades, many different types of AKE protocols have been studied in the literature (c.f. [1]). In practice, AKE has been widely used in most real-world applications (e.g., online shopping, Internet banking, webmail, SNS, remote network access). A representative example is SSL/TLS, where one can find extensive AKE protocols in the TLS 1.2 cipher suits listed in the IETF RFC 5246 [5].

In [31], Shin et al. proposed a Leakage-Resilient and Proactive Authenticated Key Exchange (for short, LRP-AKE) protocol for credential services (e.g., [17]) where a roaming client has easy-to-be-lost/stolen devices (e.g., smartphones) with some memory capacity and wants to retrieve his/her private key with the help of a credential server equipped with its database, which might be insecure against possible attacks (e.g., viruses, hackers, insider attacks). The retrieved private key is used in the conventional PKI (Public-Key Infrastructures). However, the LRP-AKE protocol itself does not depend on PKI and TPM/TPM (Tamper-Proof/Resistant Modules). Please, refer to [31] for more details.

Compared to the previous password-enabled PKI [12], [23], [28], [33] and two-factor authentication [6], [7], [16], [18], the LRP-AKE protocol provides not only a higher level of security against leakage of stored secrets but also secrecy of private key with respect to the involving server. Also, Shin et al. proved that the LRP-AKE protocol is secure in the random oracle model [3] under the computational Diffie-Hellman problem. Note that the existing widely-deployed two-factor authentication [6], [7], [16], [18] are insecure against password list attacks, Phishing attacks, and offline dictionary attacks after leakage of stored secrets from the server side because these are just a combination of password authentication and 2nd-factor (e.g., OTP/PIN/SMS) authentication. Very recently, the LRP-AKE protocol has been standardized in ISO/IEC 11770-4:2017/Amd 2:2021 [10].

1.1 Our Contributions

In this paper, we discuss what would be a problem in the security proof of the LRP-AKE protocol and its possible solutions. First, we show that the “simplified” security proof [31] for the LRP-AKE protocol cannot be applied to the entire protocol because an adversary can easily break the semantic security of session keys. Then, we propose a modified LRP-AKE protocol that has a simple and effective measure to the problem, and formally prove its AKE security (in Definition 1) and mutual authentication (in Definition 2) for the entire modified LRP-AKE protocol.

Also, we show several extensions of the (modified) LRP-AKE protocol including 1) synchronization issue between the client and server’s stored secrets; 2) randomized ID for the provision of client’s privacy; and 3) a solution to preventing server compromise-impersonation attacks. After describing our implementation with the elliptic curve domain parameter SECp256r1 [25] and an efficiency-measurement environment, we evaluate the performance overhead of the LRP-AKE protocol and show its test vectors. For example, the average processing time of the LRP-AKE client with the elliptic curve domain parameter SECp256r1 [25] is 142.2 ms while that of the 3-pass Diffie-Hellman client is 129.4 ms (please, refer to Sect. 8 for details). From the performance evaluation, we can confirm that the LRP-AKE protocol has almost the same efficiency as the (plain) Diffie-Hellman protocol [4] that does not provide authentication at all.
2. Notation

Here, we give some notation to be used throughout this paper. Let $\mathbb{G}$ be a finite, cyclic group of prime order $q$ and $g$ be a generator of $\mathbb{G}$ (quadratic residues modulo $p$ where $p = aq + 1$) where the Diffie-Hellman problem is hard. Let $h$ be another generator of $\mathbb{G}$ so that its discrete logarithm problem with $g$ (i.e., computing $b = \log_g h$) should be hard. Both $g$ and $h$ are given as public parameters [13]. In the aftermath, all the subsequent arithmetic operations are performed in modulo $p$ unless otherwise stated.

Let $k$ and $l$ denote the security parameters for hash functions and public-key cryptosystems, respectively. Also, let $N$ be a dictionary size of passwords. Let $\{0, 1\}^*$ denote the set of finite binary strings and $\{0, 1\}$ the set of binary strings of length $k$. If $D$ is a set, then $d \overset{R}{\leftarrow} D$ indicates the process of selecting $d$ at random and uniformly over $D$. Let $\| \|$ denote the concatenation of bit strings in $\{0, 1\}^*$. Let $\oplus$ denote the exclusive-OR operation of bit strings. The hash functions are denoted by $H_j : \{0, 1\}^* \rightarrow \{0, 1\}^{t_j}$ for $j = 1, 2, 3, 4, 5$ and 6 where $H_j$ are distinct secure one-way hash functions (e.g., SHA-2/3 [20], [21]) one another. And we denote by $\text{AE}$ a secure authentication encryption [2] (e.g., AES-GCM [29]). Let $C$ and $S$ be the identities of client and server, respectively, with each ID $\in \{0, 1\}^*$. Let $(K', \text{CertC})$ be a credential where the former is a partial secret for client’s private key $K$ associated with a public key, which is included in the latter (i.e., the client’s certificate $\text{CertC}$) with other information.

3. The LRP-AKE Protocol

In this section, we describe the LRP-AKE protocol [31] which consists of the initialization phase and $i$-th ($i \geq 1$) protocol execution phase.

3.1 Initialization

In this phase, client $C$ registers two kinds of secrets to server $S$ where one is used for authentication and generation of session keys, and the other is used for recovering the client’s private key. First, the client chooses two random numbers $s_1 \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and $K_{11} \overset{R}{\leftarrow} \{0, 1\}^l$ where $K_{11}$ is a partial secret and has the same size of his/her private key $K$. The client generates a verification data $W_1 \equiv h^{s_1}$, which is computed with secret value $s_1$ and his/her password $pw$ (i.e., $v_1 \equiv s_1 + pw \mod q$), and the other partial secret $K_{12} = K_{11} \oplus K$ from $K_{11}$ and the private key $K$. Then, client $C$ registers securely $W_1$ and $K_{12}$ along with his/her certificate $\text{CertC}$.

At the end of this phase, the client stores secret value $s_1$ and one partial secret key $K_{11}$ on devices that may happen to leak these secrets, and remembers his/her password $pw$ in mind.

3.2 $i$-th ($i \geq 1$) Protocol Execution

3.2.1 Masked Diffie-Hellman Protocol

When client $C$ wants to share a session key securely with server $S$ in the $i$-th ($i \geq 1$) protocol execution phase, he/she should recover $v_i$ by combining his/her password $pw$ with secret value $s_i$ stored on devices: $v_i \equiv s_i + pw \mod q$. The client chooses a random number $x \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and computes its Diffie-Hellman public value $X \equiv g^x$. The latter is masked in a way of the product of the public value with the verification data $W_i \equiv h^x \ast X^i \equiv X \ast W_i$. Then, the $X^i$ is sent to the server together with the client’s identity $C$ and the counter $i$. If $i$ is not a correct counter, server $S$ terminates the protocol. Otherwise, the server chooses a random number $y \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and computes its Diffie-Hellman public value $Y \equiv g^y$. The server also computes $X$ by dividing the received masked public value with the verification data: $X \equiv X^i / W_i$. From the resultant value $X$, the Diffie-Hellman key $DHK \equiv X^i$ is derived. With the $W_i$ and $DHK$, server $S$ easily generates an authenticator $V_S = H_3(\text{SID})(DHK)$, where a session identifier $\text{SID} = C||S||X^i||Y||W_i$, and sends its Diffie-Hellman public value $Y$ along with the server’s identity $S$ and the authenticator $V_S$. After computing $DHK \equiv Y^i$, client $C$ verifies the received authenticator $V_S$. If $V_S \neq H_3(\text{SID})(DHK)$, the client terminates the protocol. Otherwise, client $C$ generates his/her authenticator $V_C = H_2(\text{SID})(DHK)$ and a session key $SK = H_3(\text{SID})(DHK)$, and then sends the authenticator $V_C$. Similarly, server $S$ verifies the received authenticator $V_C$. If $V_C \neq H_2(\text{SID})(DHK)$, the server terminates the protocol. Otherwise, server $S$ generates a session key $SK = H_3(\text{SID})(DHK)$. In order to avoid so-called partition attacks [22], [34], both of client and server should check the subgroup order of $Y$ and $X^i$, respectively: $Y^i \equiv (X^i)^i \equiv 1$.

3.2.2 Recovering Private Key and Refreshing Stored Secrets

Using the session key in an authenticated encryption $\text{AE}$, server $S$ sends the credential $(K_{12}, \text{CertC})$ to client $C$ through the secure channel. Finally, the client recovers his/her private key from one partial secret (stored on own devices) and the other secret (received from the server), $K = K_{11} \oplus K_{12}$, so that a pair of private key and certificate can be used as in the usual PKI. If $K$ is not the correct private key, client $C$ terminates the protocol. Otherwise, the client stores the next counter $(i + 1)$ and newly-generated secrets $(s_{i+1}, K_{i+1})$ that are refreshed from the hash of the Diffie-Hellman key and other information: $s_{i+1} \equiv s_i + H_4(\text{SID})(DHK) \mod q$ and $K_{i+1} = K_{i+1} \oplus H_5(\text{SID})(DHK)$. In the same way,

Note that all the currently-used password-based protocols require users to register their passwords securely in the initialization phase.

On the other hand, the server stores $W_1$ and $(K_{12}, \text{CertC})$ on its database. Finally, they set a counter $i$ as 1. Of course, all the other intermediate values should be deleted from the devices. This initialization is securely done only once.
server $S$ stores the next counter ($i + 1$) and refreshed secrets ($W_{i+1}, (K_{i+1}^{12}, Cert_{C})$) on its database: $W_{i+1} \equiv W_i \times \mathcal{H}(SID_{DHK})$ and $K_{i+1}^{12} = K_2 \oplus H_5(SID_{DHK})$. These stored secrets will be used for the next session between the parties without changing the client’s password. Note that the frequent change of passwords might incur the risk of password to be exposed, simply because people tends to write it down on somewhere or needs considerable efforts to remember new passwords. The hash functions $H_4$ and $H_5$ used here can be replaced by pseudo random functions to produce appropriate output sizes.

4. Security Model

Here, we introduce the security model [31], which was used to prove security of the LRP-AKE protocol.

We denote by $C$ and $S$ two parties that participate in an authenticated key exchange protocol $P$. Each of them may have several instances called oracles involved in distinct, possibly concurrent, executions of $P$. We denote $C$ (resp., $S$) instances by $C^i$ (resp., $S^i$) where $i \in \mathbb{N}$, or by $U$ in the case of any instance. For the $i$-th ($i \geq 1$) session in the LRP-AKE protocol, client $C$ remembers a low-entropy secret $pw_i$ drawn from a small dictionary of password $\mathcal{D}_{Password}$, whose cardinality is $N$, and holds another secret $s_i$ and a partial private key $K_{i1}$ on insecure devices. On the other hand, server $S$ stores a verification data $W_i$ and the client’s credential $(K_{2i}, Cert_{C})$. During the execution of the protocol, an adversary has the entire control of the network and additionally has access to the parties’ stored secrets where the latter simulates insecure devices and databases. Let us show the capability of adversary $\mathcal{A}$ each query captures:

- **Execute($C^i, S^i$):** This query models passive attacks, where the adversary gets access to honest executions of $P$ between the instances $C^i$ and $S^i$ by eavesdropping.

- **Send($U, m$):** This query models active attacks by having $\mathcal{A}$ send a message to instance $U$. The adversary $\mathcal{A}$ gets back the response $U$ generates in processing the message $m$ according to the protocol $P$. A query $\text{Send}(C^i, \text{Start})$ initializes the protocol, and thus the adversary receives the first flow message.

- **Reveal($U$):** This query handles the misuse of the session key (e.g., use in a weak symmetric-key encryption) by any instance $U$. The query is only available to $\mathcal{A}$ if the instance actually holds a session key, and the latter is released to $\mathcal{A}$.

- **Leak($U, i$):** This query handles the leakage of the $i$-th stored secrets by any instance $U$. The adversary $\mathcal{A}$ gets back the secrets ($i, s_i, K_{i1}$) or ($i, W_i, (K_{2i}, Cert_{C})$) where the former (resp., the latter) is released if the instance corresponds to $C^i$ (resp., $S^i$) and an instance of the ($i - 1$)-th session is accepted. The query is available to $\mathcal{A}$ since the stored secrets might be leaked out due to lost/stolen devices, viruses or mis-behaviors of system administrators.

- **Test($U$):** This oracle is used to see whether or not the adversary can obtain some information on the session key by giving a hint on the key. The Test-query can be asked at most once by the adversary $\mathcal{A}$ and is only available to $\mathcal{A}$ if the instance $U$ is “fresh” (see below). This query is answered as follows: One flips a (private) coin $b \in \{0, 1\}$ and forwards the corresponding session key $SK$ (Reveal($U$) would output) if $b = 1$, or a random value with the same size except the session key if $b = 0$.

We say that an instance $U$ is fresh only if an adversary $\mathcal{A}$ can not distinguish the session key $SK_b$ in a trivial way (e.g., with the Reveal($U$)-query or the Leak($S^B, *$)-query). The latter is due to the fact that in a 2-party password-based AKE protocol one cannot prove the security when an adversary gets the verification data. However, the Leak($S^B, *$)-query would be considered for security of the client’s password and private key (see Sect. 4.2 of [31]).

5. A Problem

Here, we discuss what would be a problem [26] in the security proof of the LRP-AKE protocol.

In [31], Shin et al. showed that the LRP-AKE protocol is provably secure in the random oracle model [3], and discussed security of the client’s password and private key. In order to simplify the security proof, they only considered the first two flows of the $i$-th protocol execution (i.e., unilateral authentication of $S$ to $C$) where the Leak-query is restricted to instance $C^i$ as discussed in Sect. 4.

Now, we consider all the flows of the $i$-th protocol execution of the LRP-AKE protocol. In this case, there exists a problem in the security proof because adversary $\mathcal{A}$ can distinguish the session key $SK_b$ by decrypting the ciphertext of the credential $(K_{2i}, Cert_{C})$. If $\mathcal{A}$ correctly decrypts the ciphertext with $SK_b$, then $b$ will be 1. Otherwise, $b$ will be 0.

Remark 1: The LRP-AKE protocol was renamed as “LKAM1” in the ISO/IEC 11770-4:2017/Amd 2:2021 IS (International Standard) [10]. However, this LKAM1 does not inherit the above problem since the recovering private key and certificate part of the LRP-AKE protocol [31] was excluded.

6. A Modified LRP-AKE Protocol

In this section, we propose a modified LRP-AKE protocol that has a simple and effective measure to the problem discussed in Sect. 5, and then formally prove its security for the entire modified LRP-AKE protocol.

6.1 The Protocol

The problem in Sect. 5 is due to the fact that the session key is directly used for an authenticated encryption in the recovering private key and certificate part (see Sect. 3.2.2)
of the LRP-AKE protocol [31].

In the modified LRP-AKE protocol, we slightly change only the recovering private key and certificate part (Sect. 3.2.2) as follows: Instead of using the session key $SK$, both of client $C$ and server $S$ generate an encryption key $EK = H_6(SID || DHK)$ which will be used for an authenticated encryption $AE$ in order to encrypt/decrypt the credential $(K_{i+2}, Cert_C)$. Since we are working in the random oracle model, this modification is the simplest solution anyway. A graphical description of the modified LRP-AKE protocol can be found in Fig. 1 and Fig. 2.

6.2 Security Definitions and Assumption

The adversary $\mathcal{A}$ is provided with random coin tosses, some oracles and then is allowed to invoke any number of queries as described in Sect. 4, in any order. The aim of the adversary is to break the privacy of the session key (a.k.a., semantic security) or authentication of the parties in the context of executing $P$. 

![Fig. 1](image1.png)

**Fig. 1** The initialization phase of the (modified) LRP-AKE protocol where the enclosed values in the rectangle represent stored secrets of client and server, respectively.

![Fig. 2](image2.png)

**Fig. 2** The $i$-th ($i \geq 1$) protocol execution phase of the modified LRP-AKE protocol where $SID = C||S||I||X' || F|| W_i$ plays a role of session identifier and the enclosed values in the rectangle represent stored secrets of client and server, respectively.
Definition 1: (AKE Security) The AKE security is defined by the game Game\textsuperscript{ake}(A, P), in which the ultimate goal of the adversary is to guess the bit b involved in the Test-query by outputting this guess b'. We denote the AKE advantage, by \( \text{Adv}_P^\text{ake}(A) = 2 \Pr[b = b'] - 1 \), as the probability that \( A \) can correctly guess the value of b. The protocol \( P \) is said to be \((t, \varepsilon)\)-AKE-secure if \( A \)'s advantage is smaller than \( \varepsilon \) for any adversary \( A \) running time \( t \).

Definition 2: (MA) Another goal is to consider mutual authentication of \( C \) (C-auth) and \( S \) (S-auth) wherein the adversary impersonates either party. We denote by Succ\textsuperscript{ma}_P(A) the probability that \( A \) successfully impersonates an \( C \) instance (resp., an \( S \) instance) in an execution of \( P \), which means that \( S \) (resp., \( C \)) agrees on a key while the latter is shared with no instance of \( C \) (resp., \( S \)). A protocol \( P \) is said to be \((t, \varepsilon)\)-MA-secure if \( A \)'s success probability for breaking either \( C \)-auth or \( S \)-auth is smaller than \( \varepsilon \) for any adversary \( A \) running time \( t \).

Also, there is a computational assumption on which the modified LRP-AKE protocol is based.

Assumption 1: (CDH) A \((t, \varepsilon)\)-CDH\(_{g, \mathbb{G}} \) attacker, in a finite cyclic group \( \mathbb{G} \) of prime order \( q \) with \( g \) as a generator, is a probabilistic machine \( B \) running in time \( t \) such that its success probability Succ\textsuperscript{cdh}_\text{P}(\mathbb{G}, \mathbb{G})\((t)\), given random elements \( g^x \) and \( g^y \) to output \( g^{xy} \), is greater than \( \varepsilon \). We denote by Succ\textsuperscript{cdh}_\text{P}(\mathbb{G}, \mathbb{G})\((t)\) the maximal success probability over every adversaries running within time \( t \). The CDH assumption states that Succ\textsuperscript{cdh}_\text{P}(\mathbb{G}, \mathbb{G})\((t)\) \( \leq \varepsilon \) for any \( t/\varepsilon \) not too large.

### 6.3 Security Proof

Here, we show that the modified LRP-AKE protocol of Fig. 2 is provably secure in the random oracle model [3]. In other words, the modified LRP-AKE protocol distributes session keys that are semantically-secure and provides mutual authentication.

Theorem 1: Let \( P \) be the modified LRP-AKE protocol of Fig. 2 where passwords are chosen from a dictionary of size \( N \). For any adversary \( A \) within a polynomial time \( t \), with less than \( q_{\text{se}} \) active interactions with the parties (Send-queries), \( q_{\text{ex}} \) passive eavesdroppings (Execute-queries) and asking \( q_{\text{H}} \) hash queries to any \( H_j \), \( \text{Adv}_P(A) \leq 4\varepsilon \) and Succ\textsuperscript{cdh}_\text{P}(\mathbb{G}, \mathbb{G})\((t)\) \( \leq \varepsilon \), with \( \varepsilon \) upper-bounded by

\[
\frac{3q_{\text{se}}}{N} + 7q_{\text{H}}^2 \times \text{Succ}\textsuperscript{cdh}(t + 3\tau_c) + \frac{q_{\text{se}} \times q_{\text{se}}}{2\varepsilon_1} + \frac{q_{\text{se}} \times q_{\text{se}}}{2\varepsilon_2} + \frac{8q_{\text{se}} + 2q_{\text{ex}} + (q_{\text{ex}} + q_{\text{se}})^2}{2q},
\]

where \( k_1 \) (resp., \( k_2 \)) is the output length of \( H_1 \) (resp., \( H_2 \)) and \( \tau_c \) denotes the computational time for an exponentiation in \( \mathbb{G} \).

This theorem indicates that the modified LRP-AKE protocol is secure against offline dictionary attacks since the advantage of the adversary essentially grows with the ratio of interactions to the number of passwords. Note that in order to perform online dictionary attacks successively the adversary should obtain the client’s stored secrets every time.

**Proof 1:** In this proof, we define a sequence of games starting at the real protocol \( G_0 \) and ending up at \( G_5 \) where we bound the probability of each event by using Shoup’s difference lemma [27]. For clarity, we denote by Event\( _n \) an event Event considered in Game \( G_n \).

**Game \( G_0 \):** This is the real protocol in the random oracle model. We are interested in the following two events:

- \( S_0 \) which occurs if the adversary correctly guesses the bit \( b \) involved in the Test-query;
- \( A_0 \) which occurs if an instance \( C^x \) (resp., \( S^y \)) accepts with no partner instance \( S^y \) (resp., \( C^x \)) with the same transcript ((\( C, i, X^j \)), \( S, Y, V_5 \), \( V_C \)).

\[
\text{Adv}_P^\text{ake}(A) = 2 \Pr[S_0] - 1, \quad \text{Succ}_P^\text{ma}(A) = \Pr[A_0].
\]

(2)

In any game \( G_n \) below, we study the event \( A_n \) and the restricted event \( \text{SwA}_n = S_n \land \neg A_n \).

**Game \( G_1 \):** In this game, we simulate the hash oracles \( H_j \), but as well additional hash functions, for \( j = 1, 2, 3, 4, 5, 6, H'_j : \{0, 1\}^* \rightarrow \{0, 1\}^{k_j} \) that will appear in the game \( G_2 \) by maintaining hash lists \( \Lambda_H \) and \( \Lambda_H' \) (see below). We also simulate all the instances, as the real parties would do, for the Send-queries and for the Execute, Reveal, Leak and Test-queries (see further below). From this simulation, we can easily see that the game is perfectly indistinguishable from the real attack.

**Simulation of the hash functions: \( H_j \) oracles**

- For a hash-query \( H_j(q) \) (resp., \( H'_j(q) \)), such that a record \((j, q, r)\) appears in \( \Lambda_H \) (resp., \( \Lambda_H' \)), the answer is \( r \). Otherwise one chooses a random element \( r \leftarrow \{0, 1\}^{k_j} \), answers with it, and adds the record \((j, q, r)\) to \( \Lambda_H \) (resp., \( \Lambda_H' \)).

**Simulation of the LRP-AKE protocol**

**Send-queries to \( C \)**

We answer to the Send-queries to a C-instance as follows:

- A Send(\( C^x, \text{Start} \))-query is processed according to the following rules:
  - **Rule C1**
    - Choose random elements \((b, v_i) \sim \mathbb{Z}_q^* \), compute \( h \equiv g^b \) and \( W_i \equiv h^{v_i} \), and set \( s_i \equiv s_{i-1} + r \mod q \) where \( r \) is the reply of \( H_4 \) in the \((i-1)\)-th “accepted” instance. Also, we set \( K_i = K_{i-1} \oplus r \) where \( r \) is the reply of \( H_5 \) in the \((i-1)\)-th “accepted” instance.
  - **Rule C2**
    - Choose a random element \( \theta \sim \mathbb{Z}_q^* \), and compute \( X \equiv g^\theta \) and \( X^* \equiv X \times W_i \). Then the query is
If the instance $C'$ is in an expecting state, a query $\text{Send}(C', (S, Y, V_S))$ is processed by computing the Diffie-Hellman key, authenticators, session key and encryption key. We apply the following rules.

**Rule C3**

- Compute $DHK \equiv Y^\theta$.
- Compute the expected authenticators, session key and encryption key:
  $$V'_S = H_1(C||S||X'||Y||W_i||DHK),$$
  $$V_C = H_2(C||S||X'||Y||W_i||DHK),$$
  $$SK = H_3(C||S||X'||Y||W_i||DHK),$$
  $$EK = H_6(C||S||X'||Y||W_i||DHK).$$

If $V'_S \neq V^S$, it terminates. Otherwise the query is answered with $V_C$, and the instance goes to an expecting state.

If the instance $C'$ is in an expecting state, a query $\text{Send}(C', \text{AE}_{EK}(K_{2}, \text{Cert}_C))$ is processed by recovering the private key, and updating the stored secrets. We apply the following rules.

**Rule C5**

- Recover the private key and update the stored secrets:
  $$K = K_{i_1} \oplus K_{i_2},$$
  $$s_{i+1} \equiv s_i + H_4(C||S||X'||Y||W_i||DHK) \mod q,$$
  $$K_{i+1} = K_{i_1} \oplus H_5(C||S||X'||Y||W_i||DHK).$$

If $K$ is the correct private key, then the instance accepts. In any case, it terminates.

We answer to the $\text{Send}$-queries to a $S$-instance as follows:

- A $\text{Send}(S', (C, i, X'))$-query is processed according to the following rule:
  **Rule S1**
  - If $i$ is not correct, it terminates. Otherwise, choose a random element $\varphi \overset{R}{\leftarrow} \mathbb{Z}_q^*$ and compute $Y \equiv \varphi^\theta$. Then, the instance computes the Diffie-Hellman key and authenticator. We apply the following rules:
    - Compute the authenticator: $V^S = H_1(C||S||X'||Y||W_i||DHK)$. Then the query is answered with $(S, Y, V_S)$, and the instance goes to an expecting state.
    - If the instance $S'$ is in an expecting state, a query $\text{Send}(S', V_C)$ is processed by computing the alleged authenticator, session key and encryption key, and updating the stored secrets. We apply the following rule.
      **Rule S4**
      - Compute the expected authenticator, session key and encryption key:
        $$V'_C = H_2(C||S||X'||Y||W_i||DHK),$$
        $$SK = H_3(C||S||X'||Y||W_i||DHK),$$
        $$EK = H_6(C||S||X'||Y||W_i||DHK).$$

Update the stored secrets:
$$W_{i+1} \equiv W_i \times h^{H_4(C||S||X'||Y||W_i||DHK)},$$
$$K_{i+1} = K_{i_1} \oplus H_5(C||S||X'||Y||W_i||DHK),$$
If $V'_C \neq V_C$, it terminates. Otherwise, the query is answered with $\text{AE}_{EK}(K_{2}, \text{Cert}_C)$ and the instance accepts.

**Other queries**

- An $\text{Execute}(C', S')$-query is processed using successively the above simulations of the $\text{Send}$-queries: $(C, i, X') \leftarrow \text{Send}(C', \text{Start}), (S, Y, V_S) \leftarrow \text{Send}(S', (C, i, X')), V_C \leftarrow \text{Send}(C', (S, Y, V_S)), \text{AE}_{EK}(K_{2}, \text{Cert}_C) \leftarrow \text{Send}(S', V_C),$ and then outputting the transcript $((C, i, X'), (S, Y, V_S), V_C, \text{AE}_{EK}(K_{2}, \text{Cert}_C)).$

- A $\text{Reveal}(U)$-query returns the session key $SK$ computed by the instance $U$ (if the former has accepted).

- A $\text{ Leak}(C', i)$-query returns the $i$-th secret $(i, s_i, K_{i_1})$ held by the instance $C'$.

- A $\text{ Test}(U)$-query first gets $SK$ from $\text{Reveal}(U)$, and flip a coin $b$. If $b = 1$, we return the value of the session key $SK$, otherwise we return a random value drawn from $\{0, 1\}^k$.

**Game $G_2$**

For an easier analysis in the following, we cancel games in which some collisions ($\text{Coll}_2$) are unlikely to happen:

- Collisions on the partial transcripts $((C, i, X'), (S, Y, V_S))$: Any adversary tries to find out one pair $(X', Y)$, coinciding with the challenge transcript, and then obtain the corresponding session key using the $\text{Reveal}$-query. However, at least one party involves with the transcripts, and thus one of $X'$ and $Y$ is truly uniformly distributed.

The probability is bounded by the birthday paradox:
$$\Pr[\text{Coll}_2] \leq \frac{(q_{ex} + q_{se})^2}{2q}. \quad (3)$$

**Game $G_3$**

In order to make the authenticators, session key, encryption key and stored secrets unpredictable to any adversary, we compute them using the private oracles $H'_i$ (instead of $H_i$) so that the values are completely independent from the random oracles. We reach this aim by using the following rules:

- **Rule C4/C5**
  Compute the authenticators, session key and encryption key:
  $$V'_S = H'_1(C||S||X'||Y||W_i||DHK),$$
  $$V_C = H'_2(C||S||X'||Y||W_i||DHK),$$
  $$SK = H'_3(C||S||X'||Y||W_i||DHK),$$
\[ EK = H'_C (C || S || I || X^* || Y). \]

Update the stored secrets:
\[ s_{i+1} = s_i + H'_C (C || S || I || X^* || Y) \pmod{q}, \]
\[ K_{i+1} = K_{i+1} \oplus H'_C (C || S || I || X^* || Y). \]

\[ \textbf{Rule S3/S4}^{(5)} \]

Compute the authenticators, session key and encryption key:
\[ V^S = H'_C (C || S || I || X^* || Y). \]
\[ V'_S = H'_C (C || S || I || X^* || Y), \]
\[ SK = H'_C (C || S || I || X^* || Y), \]
\[ EK = H'_C (C || S || I || X^* || Y). \]

Update the stored secrets:
\[ W_{i+1} = W_i \times h^{SK} (C || S || I || X^* || Y), \]
\[ K_{i+1} = K_{i+1} \oplus H'_C (C || S || I || X^* || Y). \]

Since we do no longer need to compute the value \( DHK \), we can simplify the following rules:

\[ \textbf{Rule C3/S2}^{(4)} \]

Do nothing.

Finally, the mask \( W_i \) is not used anymore either so that we can also simplify the generation of \( X^* \) using the group property of \( G \).

\[ \textbf{Rule C2}^{(3)} \]

Choose a random element \( x \leftarrow \mathbb{Z}_q^* \) and compute \( X^* = g^x \).

The games \( G_3 \) and \( G_2 \) are indistinguishable unless some specific hash queries are asked, denoted by event \( \text{AskH}_1 = \text{AskH}_1 \text{ Passive}_1 \) and \( \text{AskH}_2 = \text{AskH}_2 \text{ Passive}_2 \) and \( \text{AskH}_4 = \text{AskH}_4 \text{ Passive}_4 \).\]

- \( \text{AskH}_1 \text{ Passive}_1 \): The transcript \( ((C, i, X^*), (S, Y)) \) comes from an execution between instances of \( C \) and \( S \) (Execute-queries or forward of Send-queries, relay of part of them). This means that both \( X^* \) and \( Y \) have been simulated;
- \( \text{AskH}_1 \text{ Passive}_2 \): The execution involved an instance of \( C \), but \( Y \) has not been sent by any instance of \( S \). This means that \( X^* \) has been simulated, but \( Y \) has been produced by the adversary;
- \( \text{AskH}_1 \text{ Passive}_3 \): The execution involved an instance of \( S \), but \( X^* \) has not been sent by any instance of \( C \). This means that \( Y \) has been simulated, but \( X^* \) has been produced by the adversary.

\[ \textbf{Game G_1} \]: In order to evaluate the above events, we introduce a random Diffie-Hellman instance \( (P, Q) \) where both \( P \) and \( Q \) are generators of \( G \). (Otherwise, the Diffie-Hellman problem is easy.) We first modify the simulation of \( C \), involving the element \( Q \).

\[ \textbf{Rule C1}^{(4)} \]

Choose a random element \( w \leftarrow \mathbb{Z}_q^* \) and compute \( W_i = h^{\text{SK}} (C || S || I || X^* || Y) \pmod{q}. \)

By the isomorphic property from \( G \) to \( G \), the new \( W_i \) is perfectly indistinguishable from before since there exists a unique discrete logarithm for \( W_i \). We also introduce the other part \( P \) of the Diffie-Hellman instance in the simulation of \( S \).

\[ \textbf{Rule S1}^{(4)} \]

If \( i \) is not correct, it terminates. Otherwise, choose a random element \( y \leftarrow \mathbb{Z}_q^* \) and compute \( Y = P^y \).

Note that we excluded the cases \( W_i = 1 \) and \( Y = 1 \). By Shoup’s difference lemma, the probability is upper-bounded by a random guess for \( Y \):

\[ |\text{Pr} \left[ \text{AskH}_1 \right] - \text{Pr} \left[ \text{AskH}_2 \right]| \leq \frac{q_{\text{se}} + q_{\text{ex}}}{q}. \]

\[ \textbf{Game G_2} \]: It is now possible to evaluate the probability of the event \( \text{AskH} \) (or more precisely, the sub-cases). Indeed, one can see that the password is never used during the simulation. It does not need to be chosen in advance, but at the very end only. Then, an information-theoretic analysis can be done which simply uses cardinalities of some sets.

To this aim, we first cancel a few more games, wherein for some pairs \((X^*, Y) \in G^2 \), involved in a communication between an instance \( S \) and either the adversary
or an instance $C$, there are two distinct elements $W_j$ such that the tuple $(i, X^*, Y, W_i, \text{CDH}_{g, \alpha}(X^*/W_i, Y))$ is in $\Lambda_H$ (which event is denoted CollH$_3$). By Shoup’s difference lemma, the probability is upper-bounded by $Pr[\text{CollH}_3]$:  

$$|Pr[\text{AskH}_3] - Pr[\text{AskH}_4]| \leq Pr[\text{CollH}_3].$$  

(6)

The event CollH$_3$ can be upper-bounded, granted the following lemma:

**Lemma 1:** If for any pair $(X^*, Y) \in \mathbb{G}^2$, involved in a communication with an instance $S'$, there are two elements $W_0$ and $W_i$ such that $(i, X^*, Y, W_{ij}, Z_{ij} = \text{CDH}_{g, \alpha}(X^*/W_{ij}, Y))$ is in $\Lambda_H$, one can solve the computational Diffie-Hellman problem:

$$Pr[\text{CollH}_3] \leq q_1^2 \times \text{Succ}_{g, \alpha}^{\text{cdh}}(t + \tau_e).$$  

(7)

**Proof.** We prove this lemma by showing the reduction to the CDH problem when event CollH$_3$ happens. We assume that there exist $(X^*, Y \equiv P^\psi) \in \mathbb{G}^2$ involved in a communication with an instance $S'$, and two elements $W_0 \equiv Q^{\psi}$ and $W_i \equiv Q^{\eta_n}$ such that the tuple $(i, X^*, Y, W_{ij}, Z_{ij} \equiv \text{CDH}_{g, \alpha}(X^*/W_{ij}, Y))$ is in $\Lambda_H$, for $j = 0, 1$. Then,

$$Z_{ij} = \text{CDH}_{g, \alpha}(X^*/W_{ij}, Y) = \text{CDH}_{g, \alpha}(X^*/Q^{\eta_n}, Y) = \text{CDH}_{g, \alpha}(X^*, Y) \times \text{CDH}_{g, \alpha}(Q, Y)^{-\eta_n} = \text{CDH}_{g, \alpha}(X^*, Y) \times \text{CDH}_{g, \alpha}(P, Q)^{-\eta_n}.$$  

(8)

As a consequence, $Z_{ij}/Z_0 = \text{CDH}_{g, \alpha}(P, Q)^{\psi_0(\eta_n-1)}$, and thus $\text{CDH}_{g, \alpha}(P, Q) = (Z_{ij}/Z_0)^\psi$ where $\psi$ is the inverse of $y(\psi_0 - \psi_1)$ in $\mathbb{Z}_q^*$. The latter exists since $W_0 \neq W_i$ and $y \neq 0$. By guessing the two queries asked to the H$_j$, one concludes the proof. □

In order to conclude the proof, let us study separately the three sub-cases of AskH1, and then AskH2w1, AskH3w12, AskH4w123, AskH5w1234 and AskH6w12345 (keeping in mind the absence of several kinds of collisions for partial transcripts, and for $W_i$ in H-queries):

- **AskH1-Passive:** About the passive transcripts (in which both $X^*$ and $Y$ have been simulated), one can state the following lemma:

**Lemma 2:** If for any pair $(X^*, Y) \in \mathbb{G}^2$, involved in a passive transcript, there is an element $W_i$ such that $(i, X^*, Y, W_i, Z_i = \text{CDH}_{g, \alpha}(X^*/W_i, Y))$ is in $\Lambda_H$, one can solve the computational Diffie-Hellman problem:

$$Pr[\text{AskH1-Passive}_3] \leq q_H \times \text{Succ}_{g, \alpha}^{\text{cdh}}(t + 2\tau_e).$$  

(9)

**Proof.** We prove this lemma by showing the reduction to the CDH problem when event AskH1-Passive$_5$ happens. We assume that there exist $(X^* \equiv g^a, Y \equiv P^\psi) \in \mathbb{G}^2$ involved in a passive transcript and $W_i = Q^\psi$, such that the tuple $(i, X^*, Y, W_i, Z_i) = \text{CDH}_{g, \alpha}(X^*/W_i, Y)$ is in $\Lambda_H$. As above,

$$Z_i = \text{CDH}_{g, \alpha}(X^*, Y) \times \text{CDH}_{g, \alpha}(Q, Y)^{-\psi} = P^{\psi y} \times \text{CDH}_{g, \alpha}(P, Q)^{-\psi y}.$$  

(10)

As a consequence, $\text{CDH}_{g, \alpha}(P, Q) = (Z/P^\psi)^\psi$ where $\psi$ is the inverse of $-y_1 = \psi_1$ in $\mathbb{Z}_q^*$. The latter exists since we have excluded the cases where $y = 0$ or $v_i = 0$. By guessing the query asked to the H$_j$, one concludes the proof.

- **AskH1-WithC:** This corresponds to an attack where the adversary tries to impersonate $S$ to $C$. But each authenticator sent by the adversary (who may have obtained the secret $s_i$ in the Leak-query, denoted by event Leak$_3$) has been computed with at most one $W_i$ value:

$$Pr[\text{AskH1-WithC}_3] = Pr[\text{AskH1-WithC}_3|\text{Leak}_3] + Pr[\text{AskH1-WithC}_3|\text{Leak}_3] \leq \frac{q_{se}}{q} = \frac{q_{se}}{N}.$$  

(11)

- **AskH1-WithS:** The Lemma 1, applied to games where the event CollH$_3$ did not happen, states that for each pair $(X^*, Y)$ involved in a transcript with an instance $S'$, there is at most one element $W_i$ such that for $W_i = Q^\psi$, the corresponding tuple is in $\Lambda_S$: The probability for the adversary (who may have obtained the secret $s_i$ in the Leak-query) over a random password is as above:

$$Pr[\text{AskH1-WithS}_3] \leq \frac{q_{se}}{q} + \frac{q_{se}}{N}.$$  

(12)

About AskH2w1 (when the above three events did not happen), it means that only executions with an instance of $S$ (and either $C$ or the adversary) may lead to acceptance. Exactly the same analysis as for AskH1-Passive and AskH1-WithS leads to

$$Pr[\text{AskH2w1}_3] \leq \frac{q_{se}}{q} + \frac{q_{se}}{N} + q_H \times \text{Succ}_{g, \alpha}^{\text{cdh}}(t + 2\tau_e).$$  

(13)

Also, we can have the same analysis about AskH3w12, AskH4w123, AskH5w1234 and AskH6w12345 since only executions involving instances of $S$ and $C$ may lead to acceptance. Thus,

$$Pr[\text{AskH3w12}_3] + Pr[\text{AskH4w123}_3] + Pr[\text{AskH5w1234}_3] + Pr[\text{AskH6w12345}_3] \leq 4q_H + \text{Succ}_{g, \alpha}^{\text{cdh}}(t + 2\tau_e).$$  

(14)

As a conclusion, we get an upper-bound for the probability of AskH3 by combining all the cases:
These extensions, of course, can be applied to the modified LRP-AKE protocol.

7. Several Extensions

In this section, we discuss several extensions of the LRP-AKE protocol (c.f. Sect. 6 of [32]). These extensions, of course, can be applied to the modified LRP-AKE protocol.

7.1 Synchronization

An important issue in the LRP-AKE protocol may be how to synchronize the respective stored secrets \((s_i, K_{i1})\) and \((W_i, K_{i2})\) of client C and server S. If this synchronization is broken, the LRP-AKE protocol terminates with an error. This indicates that the update-timing of the LRP-AKE protocol should be considered more carefully.

Let us suppose that the last message (i.e., the encrypted credential) of the LRP-AKE protocol did not reach to client C. It is inevitable in the AKE protocols that one party, who sent the last message, cannot confirm whether the other party really received the message. In such a situation, client C also did not know whether the third message \(V_c\) correctly reached to server S. If that is the case, the server stores either \((W_i, K_{i2})\) (if it did not receive the third message) or \((W_{(i+1)}, K_{(i+1)2})\) (if it received the third message). On the other hand, the client just keeps \((s_{i}, K_{i1})\) because he/she did not receive the last message. A simple way for synchronization is to change the updating procedure on the client side in Sect. 3.2.2 as follows: 1) Client C computes \((s_{i+1}, K_{(i+1)1})\) as usual, and 2) If C received the last message and checked its correctness of the private key, he/she stores \((s_{i+1}, K_{(i+1)1})\) and outputs accept; Otherwise, he/she stores two pairs \((s_i, K_{i1})\) and \((s_{(i+1)}, K_{(i+1)1})\), and terminates the protocol. In any case, client C and server S can run the LRP-AKE protocol with a pair of either \((s_i, K_{i1})\) and \((W_i, K_{i2})\) or \((s_{(i+1)}, K_{(i+1)1})\) and \((W_{(i+1)}, K_{(i+1)2})\).

If an attacker obtains the client’s stored secrets (i.e., two pairs \((s_i, K_{i1})\) and \((s_{(i+1)}, K_{(i+1)1}))\), the only information the attacker can get are hashed values \(H_4(SID)(DHK)\) and \(H_4(SID)(DHK)\). As it is clear, these values are useless in attacking the LRP-AKE protocol. Also, online dictionary attacks for these two pairs are not possible because server S holds either \((W_i, K_{i2})\) or \((W_{(i+1)}, K_{(i+1)2})\).

7.2 Randomized ID

Since the AKE protocols are usually used in a variety of applications, it is especially desirable to provide privacy of client. In the LRP-AKE protocol, an attacker can easily get an information about which client has accessed to which server since client’s identity \(I\) and its counter \(i\) are transmitted in clear to server S.

For the provision of client’s privacy, one can use an “randomized ID” in the LRP-AKE protocol. Specifically, the initialization phase is the same as Sect. 3.1, except that client C additionally registers a randomized ID id\(i\,\)random, chosen from \([0, 1]^n\). The id\(i\,\)random is stored, instead of the counter, on the client and server sides, respectively. In the i-th protocol execution phase of the LRP-AKE protocol, client C sends the first message (id\(i\,\)random, \(*\)) to server S and, at the end of this phase, id\(i\,\)random is updated to id\((i+1)\,\)random as in the same way for updating \((s_i, K_{i1})\) and \((W_i, K_{i2})\) (the other parts remain unchanged).

This is also effective against Denial-of-Service (DoS) or Distributed DoS (DDoS) attacks since an attacker should actively intercept the randomized ID, coming from the legitimate client. Note that these DoS/DDoS attacks have become one of the traditional tools for the purpose of exhausting a server’s memory and computational capability (e.g., by making the server to consume its computing power for modular exponentiations).

7.3 Preventing Server Compromise-Impersonation Attacks

In the LRP-AKE protocol, if the server’s stored secrets (i.e., \(W_i, (K_{i2}, Cert_C)\)) are compromised, an attacker can freely impersonate the client because the \(W_i\) can be seen as somewhat ‘symmetric type’ information between client C and server S though the attacker cannot recover the client’s private key \(K\). This is called Server Compromise-Impersonation (SCI) attacks in the client-server setting where an attacker obtains the sever’s secret information and then tries to impersonate the client, not the server.

A solution to preventing SCI attacks is to use ‘asymmetric type’ information between the client and the server as proposed in [9], [15]. Although a direct application of their method[9] leads to security against SCI attacks, we describe a different way to integrate any secure signature
schemes (in the sense of [8]) to the LRP-AKE protocol. Specifically, the initialization phase is the same as Sect. 3.1 except that client C generates a signature/verification key pair \((\text{Sig}K, \text{Ver}K) \leftarrow \text{SigKG}(1^t)\), where \text{SigKG} is a key generation algorithm and \(t\) is the security parameter, and registers the \text{Ver}K to server \(S\). The \text{Sig}K and \text{Ver}K are stored on the client and server sides, respectively. The parts to be changed in Sect. 3.2.1 are as follows: 1) Client C generates a signature \(\text{Sig}_C = \text{Sign}((\text{Sig}K; V_C)\), where \text{Sign} is a signature generation algorithm and \(V_C\) corresponds to a message to be signed, and sends \(\text{Sig}_C\) instead of \(V_C\); and 2) Correspondingly, server \(S\) verifies the \(\text{Sig}_C\) by checking \(\text{Verify}(\text{Ver}K; \text{Sig}_C)\), where \text{Verify} is a verification algorithm and outputs 0/1. If the \(\text{Sig}_C\) is a valid signature for \(V_C\) (i.e., \(\text{Verify}(\text{Ver}K; \text{Sig}_C) = 1\)), server \(S\) proceeds to the next step. Otherwise, it terminates the protocol. Note that compromise of the server’s stored secrets does not lead to offline dictionary attacks on the password in the LRP-AKE protocol.

8. Performance Evaluation

Here, we evaluate the performance overhead of the LRP-AKE protocol, and show its test vectors.

To this end, we implemented the LRP-AKE protocol with the elliptic curve domain parameters [25]. In order to compare with the Diffie-Hellman (key exchange) protocol [4], the recovering private key and certificate part in Sect. 3.2.2 is omitted in our implementation. For a fair comparison, we also implemented the 3-pass Diffie-Hellman protocol accompanied by key confirmation. As well known, the (3-pass) Diffie-Hellman protocol itself is not secure against active attacks.

As an efficiency-measurement environment, the LRP-AKE server is running on Ubuntu 64-bit virtual machine (Intel® 64bit Core™ i7-6500U 2.50 GHz CPU with 8 GB RAM) and the LRP-AKE client running on Raspberry Pi 2 Model B [24]. The LRP-AKE server and client are connected through 100 Mbps Ethernet.

Specifically, we measured processing times (unit: ms) of the LRP-AKE client and the 3-pass Diffie-Hellman client 15 times by the real time of ‘time’ command for each elliptic curve domain parameter [25]. These results are summarized in Table 1, and its comparative graph can be found in Fig. 3. For example, the average processing time of the LRP-AKE client with the elliptic curve domain parameter SECp256r1 [25] is 142.2 ms while that of the 3-pass Diffie-

| EC domain parameters [25] | Average | 1st | 2nd | 3rd | 4th | 5th | 6th | 7th | 8th | 9th | 10th | 11th | 12th | 13th | 14th | 15th |
|---------------------------|---------|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|------|------|------|------|-----|
| SECp160r1 (LRP-AKE)      | 126.3   | 114 | 121 | 118 | 127 | 124 | 133 | 125 | 131 | 126 | 132  | 121  | 129  | 124  | 121  | 148 |
| SECp192r1 (DHKE)         | 127.5   | 110 | 112 | 112 | 112 | 112 | 112 | 112 | 112 | 112 | 112  | 112  | 112  | 112  | 112  | 111 |
| SECp224r1                | 169.5   | 150 | 151 | 150 | 150 | 150 | 150 | 150 | 150 | 150 | 150  | 150  | 150  | 150  | 150  | 150  |
| SECp256r1                | 142.2   | 134 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132 | 132  | 132  | 132  | 132  | 132  | 132  |
| SECp384r1                | 147.8   | 130 | 150 | 150 | 178 | 145 | 139 | 148 | 147 | 156 | 137  | 149  | 136  | 155  | 155  | 142  |
| SECp521r1                | 211.0   | 204 | 198 | 199 | 193 | 203 | 202 | 201 | 223 | 215 | 209  | 206  | 219  | 224  | 209  | 209  |
| SECp163r2                | 136.3   | 130 | 131 | 139 | 137 | 132 | 146 | 141 | 134 | 129 | 136  | 136  | 134  | 149  | 135  | 137  |
| SECp233r1                | 123.5   | 118 | 120 | 120 | 123 | 131 | 120 | 124 | 121 | 127 | 123  | 127  | 122  | 130  | 130  | 115  |
| SECp283r1                | 167.2   | 155 | 155 | 157 | 154 | 183 | 157 | 156 | 155 | 163 | 154  | 163  | 161  | 152  | 159  | 170  |
| SECp409r1                | 261.2   | 257 | 265 | 275 | 268 | 297 | 254 | 268 | 236 | 261 | 240  | 252  | 264  | 244  | 267  | 270  |
| SECp571r1                | 488.2   | 507 | 522 | 470 | 471 | 486 | 607 | 483 | 482 | 489 | 484  | 466  | 471  | 471  | 463  | 451  |

Table 1: Processing times (unit: ms) of the LRP-AKE client on Raspberry Pi 2 Model B [24] where figures in the second line of each row represent processing times of the 3-pass Diffie-Hellman key exchange client.
Fig. 3 Comparison of processing times of the LRP-AKE client and the 3-pass Diffie-Hellman key exchange ((3P-)DHKE) client on Raspberry Pi 2 Model B [24]

### Table 2
Average operation times (unit: ns) on Raspberry Pi 2 Model B [24]

| EC domain parameters [25] | EC key pair generation | EC point multiplication | EC point addition | SHA1 | SHA224 | SHA256 | SHA384 | SHA512 |
|---------------------------|------------------------|-------------------------|-------------------|------|--------|--------|--------|--------|
| SECp160r1                 | 1113738                | 4397089                 | 41583.2           | -    | -      | -      | -      | -      |
| SECp192r1                 | 12985669               | 6211648                 | 47190.9           | -    | 103086.5 | -      | -      | -      |
| SECp224r1                 | 41108497.7             | 8550007.4               | 48743.1           | -    | 97242.9 | -      | -      | -      |
| SECp256r1                 | 7341093                | 1717354.2               | 66704.7           | -    | -      | 8450.5 | -      | -      |
| SECp384r1                 | 36787427               | 29695497                | 77569.3           | -    | -      | -      | 63381.7| -      |
| SECp521r1                 | 79498928.2             | 68261936.8              | 125746.2          | -    | -      | -      | -      | 61729.3|
| SECt163r2                 | 11515782.5             | 4883670.9               | 161270.5          | -    | 106912.8 | -      | -      | -      |
| SECt233r1                 | 15085942.2             | 8226115.6               | 206412.8          | -    | -      | -      | 86867.8| -      |
| SECt283r1                 | 22670097.1             | 15243674.3              | 262252.8          | -    | -      | -      | 55249.9| -      |
| SECt409r1                 | 37686282.5             | 29891232.1              | 387558.3          | -    | -      | -      | -      | 68419.9|
| SECt571r1                 | 78301691               | 70874813.3              | 627668.8          | -    | -      | -      | -      | 59201.2|

Hellman client is 129.4 ms. The communication overhead of the LRP-AKE protocol is 1024 bits (same as the 3-pass Diffie-Hellman protocol) where the client/server identities are not counted. In terms of the number of message exchanges, the LRP-AKE protocol requires 3 passes. From Table 1 and Fig. 3, it is clear that the LRP-AKE protocol has almost the same efficiency as the (plain) Diffie-Hellman protocol that does not provide authentication at all.

In addition, we measured average times (unit: ns) of the EC key pair generation, EC point multiplication, EC point addition, SHA1, SHA224, SHA256, SHA384, and SHA512 15 times by the real time of ‘time’ command for each elliptic curve domain parameter [25]. The library used for each operation is OpenSSL 1.1.1a. These average operation times on Raspberry Pi 2 Model B [24], whose CPU and OS are 32-bit ARM and Raspbian, respectively, can be found in Table 2. For example, the average time of SHA256 with the elliptic curve domain parameter SECp256r1 [25] is 84850.5 ns (i.e., 0.08485 ms), which is negligible in the corresponding processing time 142.2 ms of the LRP-AKE client.

In Table 3 (resp., Table 4), a test vector of the LRP-AKE protocol is listed where the elliptic curve domain parameter SECp256r1 (resp., SECt283r1) [25] and SHA256 (resp., SHA384) are used. Note that points on the elliptic curve are expressed with the compressed form having prefix 02 or 03.

### 9. Conclusions

In this paper, we pointed out a problem in the security proof of the LRP-AKE protocol [31] and then proposed a modified LRP-AKE protocol that has a simple and effective measure to the problem. Also, we formally proved its AKE security and mutual authentication for the entire modified LRP-AKE
protocol.

In addition, we discussed several extensions of the (modified) LRP-AKE protocol including 1) synchronization issue between the client’s and server’s stored secrets; 2) randomized ID for the provision of client’s privacy; and 3) a solution to preventing server compromise-impersonation at-
tacks. Finally, we evaluated the performance overhead of the LRP-AKE protocol and showed its test vectors. From the performance evaluation, we confirmed that the LRP-AKE protocol has almost the same efficiency as the (plain) Diffie-Hellman protocol [4] that does not provide authentication at all.

A possible alternative solution to the problem discussed in Sect. 5 is to adopt the authenticated and confidential channel establishment (ACCE) protocols model[11], [14], [19] where a key exchange protocol and an authenticated encryption scheme can be used overlappingly. However, in this case, one should prove the security of the LRP-AKE protocol from scratch.

Acknowledgments

We sincerely appreciate the anonymous reviewers’ constructive and valuable comments on this paper.

References

[1] C. Boyd, A. Mathuria, and D. Stebila, Protocols for Authentication and Key Establishment, Second Edition, Springer, 2020.
[2] M. Bellare and C. Namprempre, “Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm,” Journal of Cryptology, vol.21, pp.469–491, 2008.
[3] M. Bellare and P. Rogaway, “Random Oracles are Practical: A Paradigm for Designing Efficient Protocols,” Proc. ACM CCS ’93, pp.62–73, 1993.
[4] W. Diffie and M.E. Hellman, “New Directions in Cryptography,” IEEE Trans. Inf. Theory, vol.22, no.6, pp.644–654, 1976.
[5] T. Dierks and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” IETF RFC 5246, Aug. 2008. Available at https://tools.ietf.org/html/rfc5246.
[6] FIDO Alliance, “Universal 2nd Factor (U2F) Overview,” Proposed Standard, April 2017. Available at https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/FIDO-U2F-COMplete-v1.2-ps-20170411.pdf.
[7] Google Authenticator, https://en.wikipedia.org/wiki/Google_Authenticator.
[8] S. Goldwasser, S. Micali, and R.L. Rivest, “A Digital Signature Scheme Secure against Adaptive Chosen-Message Attacks,” SIAM Journal of Computing, vol.17, no.2, pp.281–308, 1988.
[9] C. Gentry, P. MacKenzie, and Z. Ramzan, “A Method for Making Password-Based Key Exchange Resilient to Server Compromise,” Proc. CRYPTO 2006, LNCS 4117, pp.149–159, Springer, 2006.
[10] ISO/IEC 11770-4:2017/Amd 2:2021, “Information technology – Security techniques – Key management – Part 4: Mechanisms based on weak secrets – Amendment 2: Leakage-resilient password-authenticated key agreement with additional stored secrets,” Feb. 2021. Available at https://www.iso.org/standard/76352.html.
[11] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, “On the Security of TLS-DHE in the Standard Model,” Proc. CRYPTO 2012, LNCS 7417, pp.273–293, Springer, 2012.
[12] T. Kwon, “Virtual Software Tokens - A Practical Way to Secure PKI Roaming,” Proc. Infrastructure Security (InfraSec), LNCS 2437, pp.288–302, Springer, 2002.
[13] W. Ladd and B. Kaduk, “SPAKE2, a PAKE,” Internet-Draft, Jan. 2021. Available at https://www.ietf.org/archive/id/draft-irtf-cfrg-spake2-18.txt.
[14] Y. Li, S. Schäge, Z. Yang, F. Kohlar, and J. Schwenk, “On the Security of the Pre-shared Key Ciphersuites of TLS,” Proc. PKC 2014, LNCS 8383, pp.669–684, Springer, 2014.
[15] P. MacKenzie, “The PAK Suite: Protocols for Password-Authenticated Key Exchange,” DIMACS Technical Report, 2002–46, 2002.
[16] Microsoft Authenticator, https://www.microsoft.com/en-us/p/microsoft-authenticator9nblggzmcj6#activetab=pivot:overviewtab.
[17] MyProxy, http://grid.ncsa.illinois.edu/myproxy/.
[18] Multi-Factor Authentication (MFA), https://dao.com/product/multi-factor-authentication-mfa.
[19] M. Manulis, D. Stebila, F. Kiefer, and N. Denham, “Secure Modular Password Authentication for the Web Using Channel Bindings,” International Journal of Information Security, vol.15, no.6, pp.597–620, 2016.
[20] NIST FIPS PUB 180-4, “Secure Hash Standard (SHS),” Aug. 2015. Available at https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.180-4.pdf.
[21] NIST FIPS PUB 202, “SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions,” Aug. 2015. Available at http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.202.pdf.
[22] S. Patel, “Number Theoretic Attacks on Secure Password Schemes,” Proc. IEEE Symposium on Security and Privacy, IEEE Computer Society, pp.236–247, 1997.
[23] R. Perlman and C. Kaufman, “Secure Password-Based Protocol for Downloading a Private Key,” Proc. Network and Distributed System Security (NDSS) Symposium 1999, Internet Security, 1999.
[24] Raspberry Pi 2 Model B, https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.
[25] Standards for Efficient Cryptography, “SEC 2: Recommended Elliptic Curve Domain Parameters (Version 2.0),” Certicom Corp., Jan. 2010. Available at http://www.secg.org/sec2-v2.pdf.
[26] S.H. Shin, “Two-factor Authentication LRP-AKE, Revisited,” Proc. 2018 International Symposium on Internet of Things & Internet of Everything, pp.990–995, IEEE Computer Society, 2018.
[27] V. Shoup, “OAEP Reconsidered,” Journal of Cryptology, vol.15, no.4, pp.223–249, 2002.
[28] R. Sandhu, M. Bellare, and R. Ganesan, “Password Enabled PKI: Virtual Smartcards vs. Virtual Soft Tokens,” Proc. 1st Annual PKI Research Workshop, pp.89–96, 2002.
[29] J. Salowey, A. Choudhury, and D. McGrew, “AES Galois Counter Mode (GCM) Cipher Suites for TLS,” IETF RFC 5288, Aug. 2008. Available at https://tools.ietf.org/html/rfc5288.
[30] S. Shin and K. Kobara, “Efficient Augmented Password-Only Authentication and Key Exchange for IKEv2,” IETF RFC 6628, June 2012. Available at https://tools.ietf.org/html/rfc6628.
[31] S.H. Shin, K. Kobara, and H. Imai, “A Secure Authenticated Key Exchange Protocol for Credential Services,” IEICE Trans. Fundamentals, vol.E91-A, no.1, pp.139–149, 2008.
[32] S.H. Shin, K. Kobara, and H. Imai, “An RSA-Based Leakage-Resilient Authenticated Key Exchange Protocol Secure against Replacement Attacks, and Its Extensions,” IEICE Trans. Fundamentals, vol.E93-A, no.6, pp.1086–1101, 2010.
[33] X. Wang, “Intrusion-Tolerant Password-Enabled PKI,” Proc. 2nd Annual PKI Research Workshop, pp.44–53, 2003.
[34] Z. Wan and S. Wang, “Cryptography of Two Password-Authenticated Key Exchange Protocols,” Proc. ACISP 2004, LNCS 3108, pp.164–175, Springer, 2004.
SeongHan Shin received his B.S. and M.S. degrees in computer science from Pukyong National University, Busan, Korea, in 2000 and 2002, respectively. In 2005, he received his Ph.D. degree in information and communication engineering, information science and technology from The University of Tokyo, Tokyo, Japan. From October 2005 to March 2006, he was a post-doctoral researcher in the Institute of Industrial Science (IIS), The University of Tokyo. In April 2006, he joined the National Institute of Industrial Science and Technology (AIST), Japan. Currently, he is a senior research scientist in Security Assurance Scheme Research Team, Cyber Physical Security Research Center (CPSEC), AIST. He received the CSS Student Paper Award in 2003, the IWS2005/WPMC’05 Best Student Paper Award in 2005, and the WISA Best Paper Premium Award in 2019. His research interests include information security, cryptography, and wireless security.