New Security Layer for OverLay Networks

Hideki Imai, SeongHan Shin, and Kazukuni Kobara

(Invited Paper)

Abstract: **After clarifying the underlying problems in a secure network storage, we introduce two important requirements, leakage-resilience and availability in higher levels respectively, for data keys that are used to protect remotely-stored data. As a main contribution of this paper, we give a new security layer for overlay networks by proposing a leakage-resilient authentication and data management system. In this system, we specifically propose a single mode and a cluster mode where the latter provides a higher level of both leakage-resilience and availability for the data key.**

Index Terms: **Availability, leakage-resilience, network storage.**

I. INTRODUCTION

Along with the advances of information processing technologies, a storage system has become an indispensable part of any whole systems. Such storage systems include direct-attached storage, network-attached storage, object storage and storage-area network. Recently, network storage (e.g., [1], [2]) has become more and more practical in the real world with which a user can store large amount of data to a remotely-located server and retrieve the data, whenever they are needed, from anywhere over insecure networks (i.e., the Internet).

The most important security issues for network storage are access control (i.e., user authentication), data confidentiality (i.e., no unauthorized access to data), data integrity (i.e., no unauthorized modification of data) and availability of data because the traditional threats to network storage are physical access to storage, access to network, authorized parties, unauthorized parties and so on. In the literature, many works (e.g., [3]–[6]) concerning with these issues have been reported for several years. In [3], Miller *et al.* proposed a secure network-attached storage system that utilizes symmetric-key encryptions and keyed hash functions on a raw disk in order to provide data confidentiality and data integrity, respectively, where the encryption key is stored on the network storage encrypted with a user's public key (of course, the user holds the corresponding private key and a key for keyed hash functions). In [4], Goh *et al.* proposed a secure remote file storage system where all file data (encrypted and signed with a user's asymmetric encryption key and signature key) are stored on the server with meta data information. In [5], Mykletun *et al.* proposed several methods to ensure data integrity by using digital signatures (condensed-RSA and BGLS [7]) in the outsourced database model where any users can make many queries to the (public) database. In [6], Heitzmann *et al.* proposed efficient checking methods of data integrity by using authenticated skip lists [8] in outsourced storage.

Actually, data confidentiality and data integrity are directly related to the following problem: How to protect the data key[1] that would be used for symmetric-key encryption/decryption algorithms and message authentication codes (or digital signatures)? On the other hand, access control is related to the problem: Which kind of authenticated key exchange protocol is used in order to retrieve the data *securely* from the remote storage server? Unfortunately, the previous works did not consider these problems at the same time.

A. Authenticated Key Exchange

Most of the network services require user authentication as well as secure channels both of which can be accomplished by using an authenticated key exchange protocol. This protocol allows the involving parties to authenticate each other and, if the authentication is finished successfully, to generate secure session keys for protecting the subsequent communications between the parties. Since authenticated key exchange is one of the important cryptographic primitives, this topic has been studied extensively so far in the cryptographic community. Some authenticated key exchange protocols, secure against active attacks (e.g., eavesdropping, messages modification, impersonation, man-in-the-middle attacks), can be found in EAP [9], [10], SSL/TLS [11], [12], IKE [13], [14] and IEEE P1363/1363.2 [15], [16].

At first sight, one may wonder why the above-mentioned problems should be considered for network storage because secure authenticated key exchange protocols are already available. The answer can be found below. Note that security of the typical authenticated key exchange protocols is based on the assumption that the keys/secrets (used for authentication) are completely secure. Here comes a natural question: What happens if these keys/secrets are leaked out to an attacker? In fact, leakages of the keys/secrets render the existing authenticated key exchange protocols to be insecure even if two/more-factor authentication is used (refer to [17]–[19] for an exclusive analysis). More seriously, such leakages are very common in the real world [20]:

- An attacker can get users' passwords by using social engineering attacks (e.g., Phishing attacks) or by using a keylogger, implanted on clients.
- Leakage of stored keys/secrets happens because mobile devices (including USB memory) are stolen or lost.

Manuscript received May 1, 2009.

The authors are with the Research Center for Information Security (RCIS), National Institute of Advanced Industrial Science and Technology (AIST), 1-18-13 Sotokanda, Chiyoda-ku, Tokyo 101-0021 Japan, email: {h-imai, seonghan.shin, k-kobara}@aist.go.jp.

The authors are also with the Chuo University, 1-13-27, Kasuga, Bunkyo-ku, Tokyo 112-8551 Japan.

---

[1] The data key can be viewed as "root key" from which any functioning keys (e.g., file encrypting key, MAC key) are derived.

- A dishonest server administer can easily obtain users' passwords.[2] This may result in a catastrophe if a user registers one (or very similar) password to many different servers. Unfortunately, that's the common practice for general users.

In order to cope with active attacks and leakage of stored secrets, we proposed several types of leakage-resilient authenticated key exchange protocols [17]–[19] where a user remembers only one password and additionally stores another secret on client while communicating with many different servers (e.g., web server, mail server, ftp). The leakage-resilient authentication protocols provide a higher level of security over the previous ones in the sense that leakage of the stored secrets from client and server does not affect its security. Note that these protocols rely on neither public-key infrastructures (PKI) nor tamper resistant modules (TRM) at all.

B. Our Motivation

Our motivation starts from the fact that, if we have to consider leakage of stored secrets from client and/or server, the security in network storage should be reconsidered from scratch because the previous works for data confidentiality and data integrity assume that client has a *secure* local storage for the keys (used in the underlying encryption and/or integrity checking schemes). An intuitive solution might be to design a secure network storage system on top of access control with the leakage-resilient authentication protocols. However, this solution does not necessarily provide a maximum security of the data.

C. Requirements for Network Storage

Here, we summarize two important requirements for network storage.

- **A higher level of security for data key:** A user's data key should be secure against active attacks as well as leakage of stored secrets from client and/or server. This requirement can be interpreted in that the user's data should be protected from untrusted networks, untrusted client's storage and untrusted server's storage.
- **Availability of data key:** A user's data key should be distributed among multiple parties so that the user can recover the key even if some parties are unavailable or physically-broken. This requirement enables the user's (confidentiality/integrity-preserving) data to be stored at any untrusted servers.

D. Our Approach

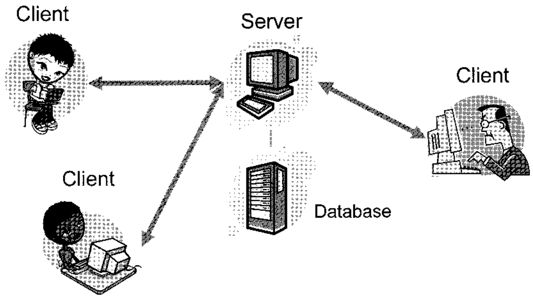


Fig. 1. Single mode.

In order to protect a user's data from various kinds of attacks, we take a novel approach by explicitly incorporating the leakage-resilient authentication protocol [19] into a data management system. As we explained in Section I-A, the leakage-resilient authentication protocol provides a higher level of security against active attacks as well as leakage of stored secrets over previous ones. The main idea is that 1) a user generates his/her data key dk, which would be used to encrypt/decrypt personal data by using symmetric-key encryption/decryption algorithms (e.g., AES) and/or to generate message authentication codes of the data, and then divide the key into two parts cdk (to be stored on client) and sdk (to be stored on server); 2) In the setup, the user registers both the partial data key sdk and the authentication secret (needed in the leakage-resilient authentication protocol) to a server; and 3) In the actual protocol execution, the user and the server mutually authenticate each other and, if the authentication is successful, share a common secret that is used to not only generate a session key but also update the current stored secrets (including the partial data keys) with new ones. With this approach, we can also achieve a higher level of security for the data key as the leakage-resilient authentication protocol [19] holds. For example, the data key remains hidden even if the stored secrets of client *and* server are leaked out with each exposed in a different time slot. See the subsequent sections for more details.

In order to provide availability of the data key, we propose a cluster mode for the leakage-resilient authentication and data management system where the data key is distributed among three parties such that any pair of (legitimate) parties can recover the key at any time. At the cost of availability, the cluster mode becomes somewhat complicated because synchronization of the three parties should be maintained at all times.

E. Organization

In Section II, we propose a single mode for the leakage-resilient authentication and data management system which guarantees a higher level of security for the data key. Section III is devoted to explaining a cluster mode that is designed to provide availability of the data key based on the single mode. Finally, concluding remarks are presented in Section IV.

II. A LEAKAGE-RESILIENT AUTHENTICATION AND DATA MANAGEMENT SYSTEM (SINGLE MODE)

In this section, we propose a single mode for the leakage-resilient authentication and data management system (see Fig. 1). Actually, this is the same scenario as considered in [19] where each pair of client and server communicate through open networks and the client (resp., the server) maintains stored secrets on memory (resp., database).

Before going into the sub-protocols, we first explain some notation that will be used throughout this paper (see Table 1). For the complete description, we fix a one-way hash function to

---

[2] In the case of hashed low-entropy password, the server administrator can find out the correct password with off-line dictionary attacks.

Table 1. Some notation.

Notation	Meaning
uid/cid/sid	user/client/server id
pcid/hpcid	index for key block, and one-time ID
pw	user's password
cs/ss	client/server authentication secret
msk	mask for RSA ciphertext
dk	user's data key
dkmsk	mask for data key
cdk/sdk	client/server partial data key where dk=cdk$\oplus$sdk$\oplus$dkmsk
(e, d, n)	RSA public key PK=(e, n) and RSA private key SK=(d, n)
RSAE(M)	RSA encryption of message M with PK
RSAD(C)	RSA decryption of ciphertext C with SK
E_k_iv(...)	symmetric-key encryption with key k and initial vector iv (e.g., AES-CBC mode)
D_k_iv(...)	symmetric-key decryption with key k and initial vector iv (e.g., AES-CBC mode)
H(...)	secure one-way hash function
HMAC(...)	keyed-hashing for message authentication code (MAC)
Random(c)	a random number is chosen from space c or c's length
a$\oplus$b	bit-wise exclusive-OR operation of a and b
a$\|$b	concatenation of a and b
a$\neq$b	a is not equal to b
reject	terminate with error

SHA-256 [21], a keyed-hashing to HMAC-SHA256 [22], and an RSA public-key encryption [23] to RSA-2048 that uses a 2048-bit RSA modulus n. In a one-way hash function, we assign the first 8-bit input as a preamble value in order to produce a different output.

A. *Overall Transition Flow*

In this subsection, we explain how a single mode for the leakage-resilient authentication and data management system works (see Figs. 2 and 3).

There are two types of setup: easy setup and usual setup. In easy setup, a server generates a disposable (short) password[3] and registers it to its database along with some public information (i.e., uid, cid and sid). Then, a pair of client and server performs the initialization protocol and the DK/SEC update protocols. In usual setup, a server generates client's stored secrets and its own secrets where the former is securely handed over to a user and the latter is stored in the database. Note that these secrets include a necessary information for the leakage-resilient authentication protocol, however, the user's password is not registered at this moment. Then, a pair of client and server performs the regular protocol and the PWD/DK update protocols. The setup is done only once and, if it is finished successfully, the client and the server are ready to perform the regular protocol at any time.

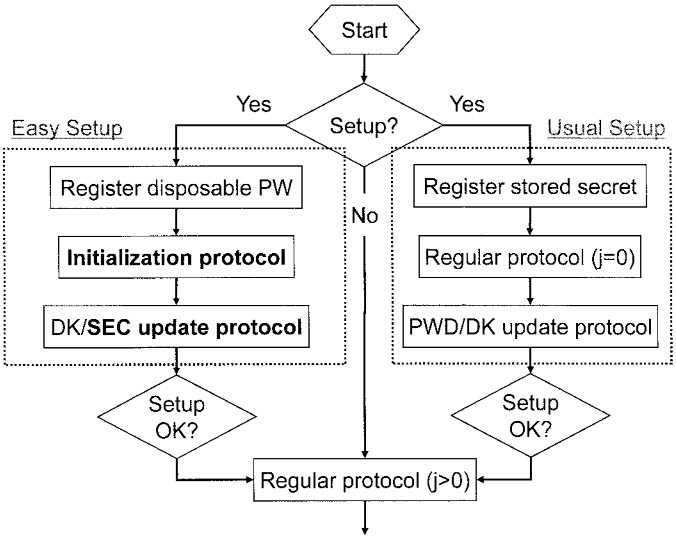


Fig. 2. The flow chart (setup).

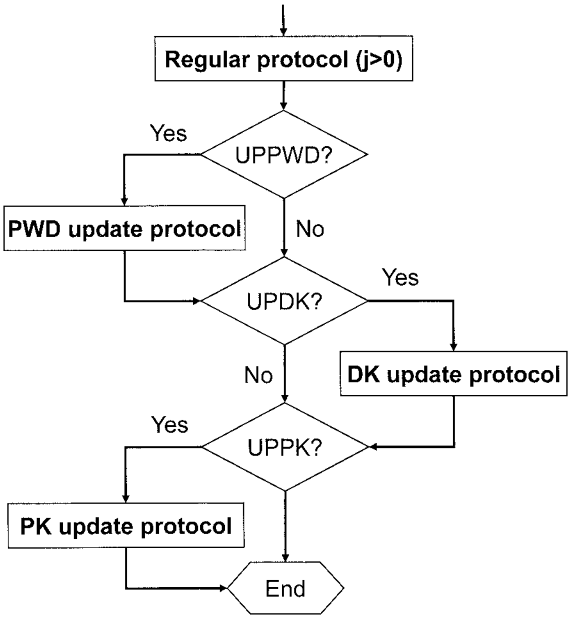
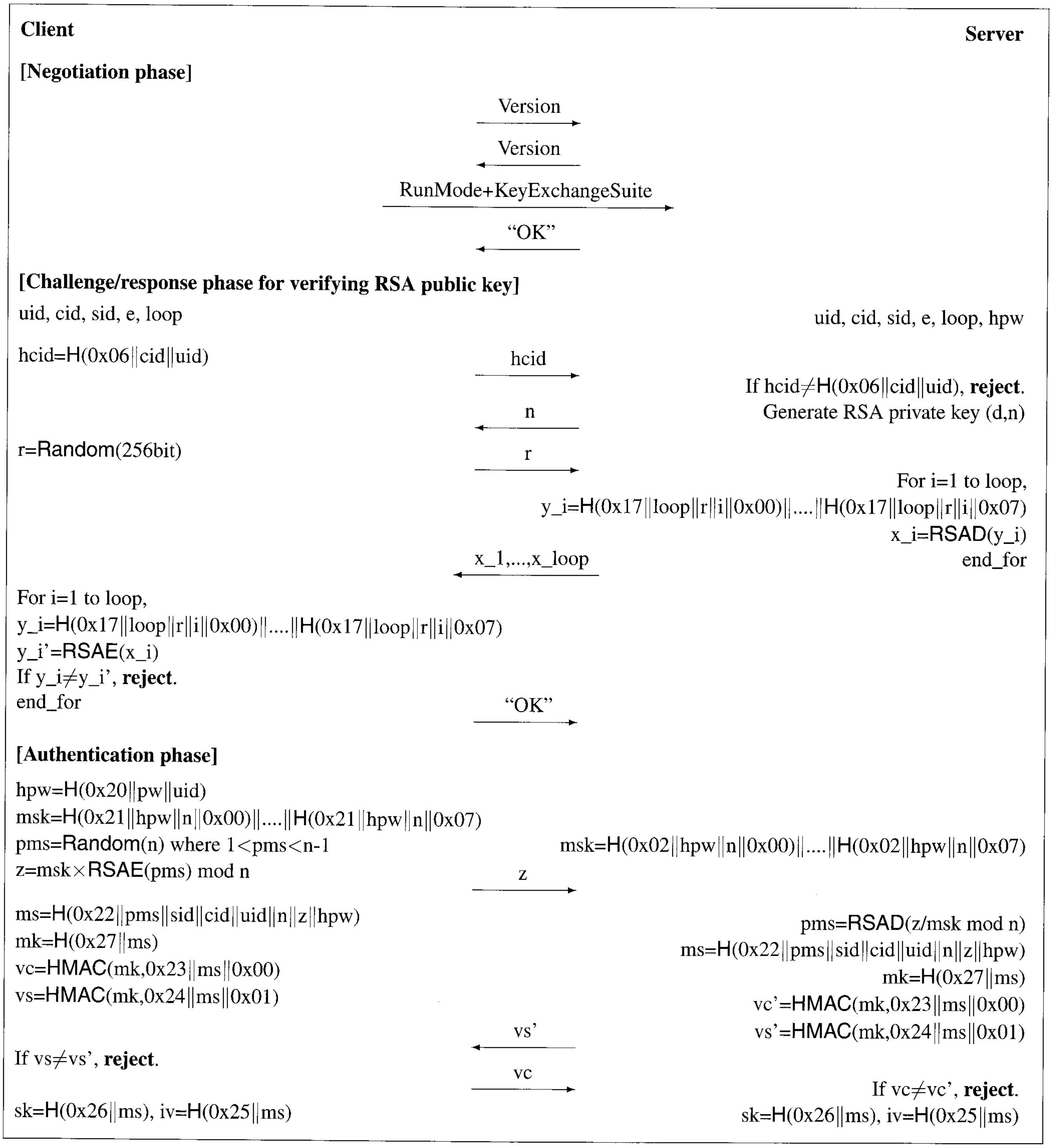


Fig. 3. The flow chart (j-th protocol execution).

After the setup, the client and the server would perform the regular protocol with the respective stored secrets. In the regular protocol, if they agree to update password/data key/PK, the corresponding update protocols (i.e., PWD/DK/PK update protocols) are followed. Note that all update protocols should be done *securely*. This is possible because messages of these update protocols are exchanged through secure channels, established between the client and the server by running the initialization and regular protocols. Specifically, if both parties authenticate each other, they can share a master secret (ms) from which a MAC key (mk) for integrity check, and a symmetric-key (sk) and an initial vector (iv) for confidentiality are generated. These shared secrets are used to realize secure channels between the client and the server. From here on, we briefly explain each subprotocol.

[3] Depending on a situation, the password can be chosen by a user.



```
Client                                                                Server
[Negotiation phase]
                                  Version  -->
                                  <--  Version
                     RunMode+KeyExchangeSuite  -->
                                  <--  "OK"

[Challenge/response phase for verifying RSA public key]
uid, cid, sid, e, loop                               uid, cid, sid, e, loop, hpw
hcid=H(0x06||cid||uid)            hcid  -->
                                                  If hcid≠H(0x06||cid||uid), reject.
                                  <--  n          Generate RSA private key (d,n)
r=Random(256bit)                  r  -->
                                                                For i=1 to loop,
                  y_i=H(0x17||loop||r||i||0x00)||....||H(0x17||loop||r||i||0x07)
                                                                   x_i=RSAD(y_i)
                                  <--  x_1,...,x_loop                    end_for
For i=1 to loop,
y_i=H(0x17||loop||r||i||0x00)||....||H(0x17||loop||r||i||0x07)
y_i'=RSAE(x_i)
If y_i≠y_i', reject.
end_for                           "OK"  -->

[Authentication phase]
hpw=H(0x20||pw||uid)
msk=H(0x21||hpw||n||0x00)||....||H(0x21||hpw||n||0x07)
pms=Random(n) where 1<pms<n-1      msk=H(0x02||hpw||n||0x00)||....||H(0x02||hpw||n||0x07)
z=msk×RSAE(pms) mod n             z  -->

ms=H(0x22||pms||sid||cid||uid||n||z||hpw)                  pms=RSAD(z/msk mod n)
mk=H(0x27||ms)                                ms=H(0x22||pms||sid||cid||uid||n||z||hpw)
vc=HMAC(mk,0x23||ms||0x00)                                           mk=H(0x27||ms)
vs=HMAC(mk,0x24||ms||0x01)                              vc'=HMAC(mk,0x23||ms||0x00)
                                  <--  vs'              vs'=HMAC(mk,0x24||ms||0x01)
If vs≠vs', reject.
                                  vc  -->                     If vc≠vc', reject.
sk=H(0x26||ms), iv=H(0x25||ms)                      sk=H(0x26||ms), iv=H(0x25||ms)
```

Fig. 4. Initialization protocol (continue to Fig. 5).

A.1 Initialization Protocol

The initialization protocol is designed for easy setup where a user just remembers a disposable password and the corresponding server stores a hashed value (hpw) of the password (see Figs. 4 and 5). This disposable password is only valid for a short period of time in order to avoid Denial-of-Service (DoS) attacks.

In the negotiation phase, a pair of client and server determines which version of the protocol and which "RunMode+KeyExchangeSuite" would be used subsequently. Here, "RunMode+KeyExchangeSuite" is fixed to the setup of single mode. In the challenge/response phase, the client verifies whether an RSA modulus n, received from the server, is correctly generated or not. Specifically, 1) the server generates "loop" number of full-domain hash (FDH) signatures [24], [25] of random number r, chosen by the client, with the RSA private key SK=(d,n) and sends them to the client; and 2) if all the signatures are verified, then the client moves to the next phase. In

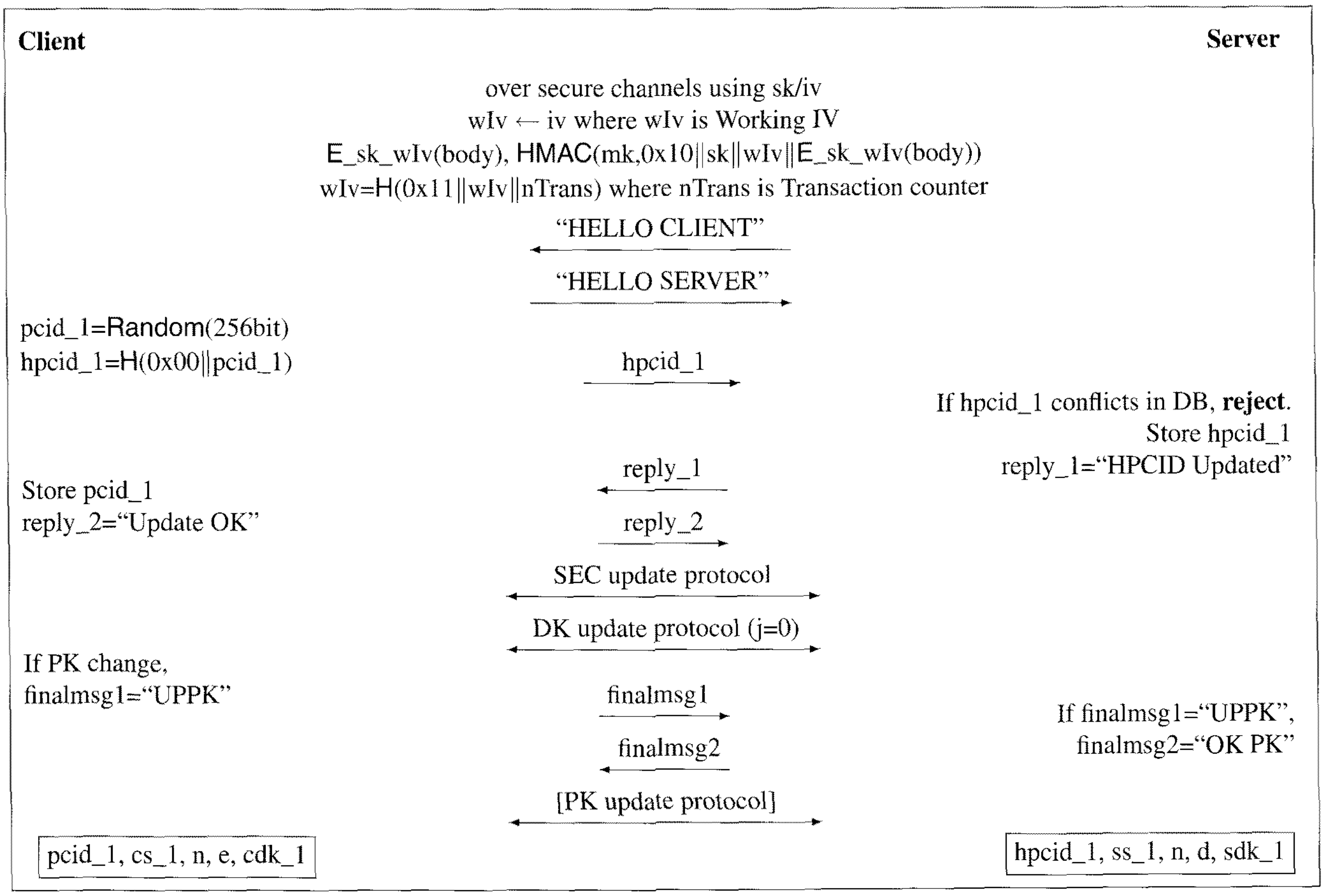


Fig. 5. Initialization protocol (continue from Fig. 4) where the enclosed values in the rectangle represent stored secrets of client and server, respectively.

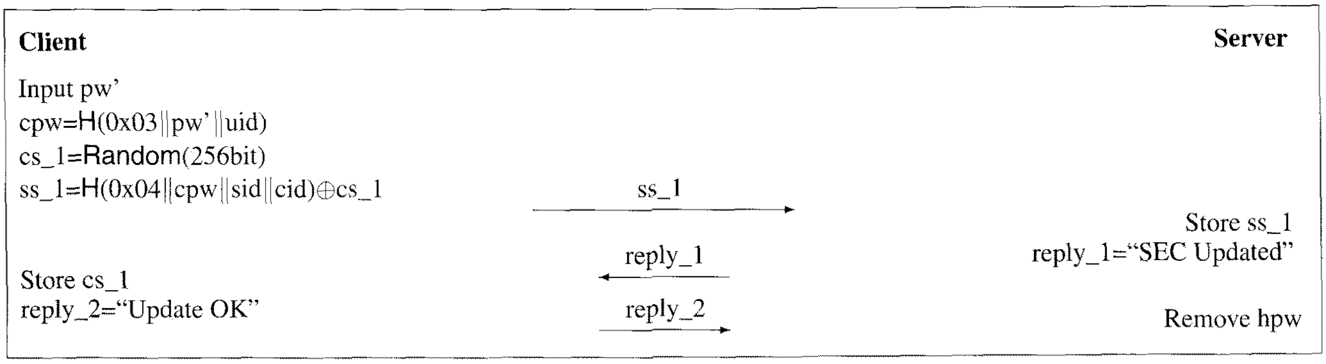
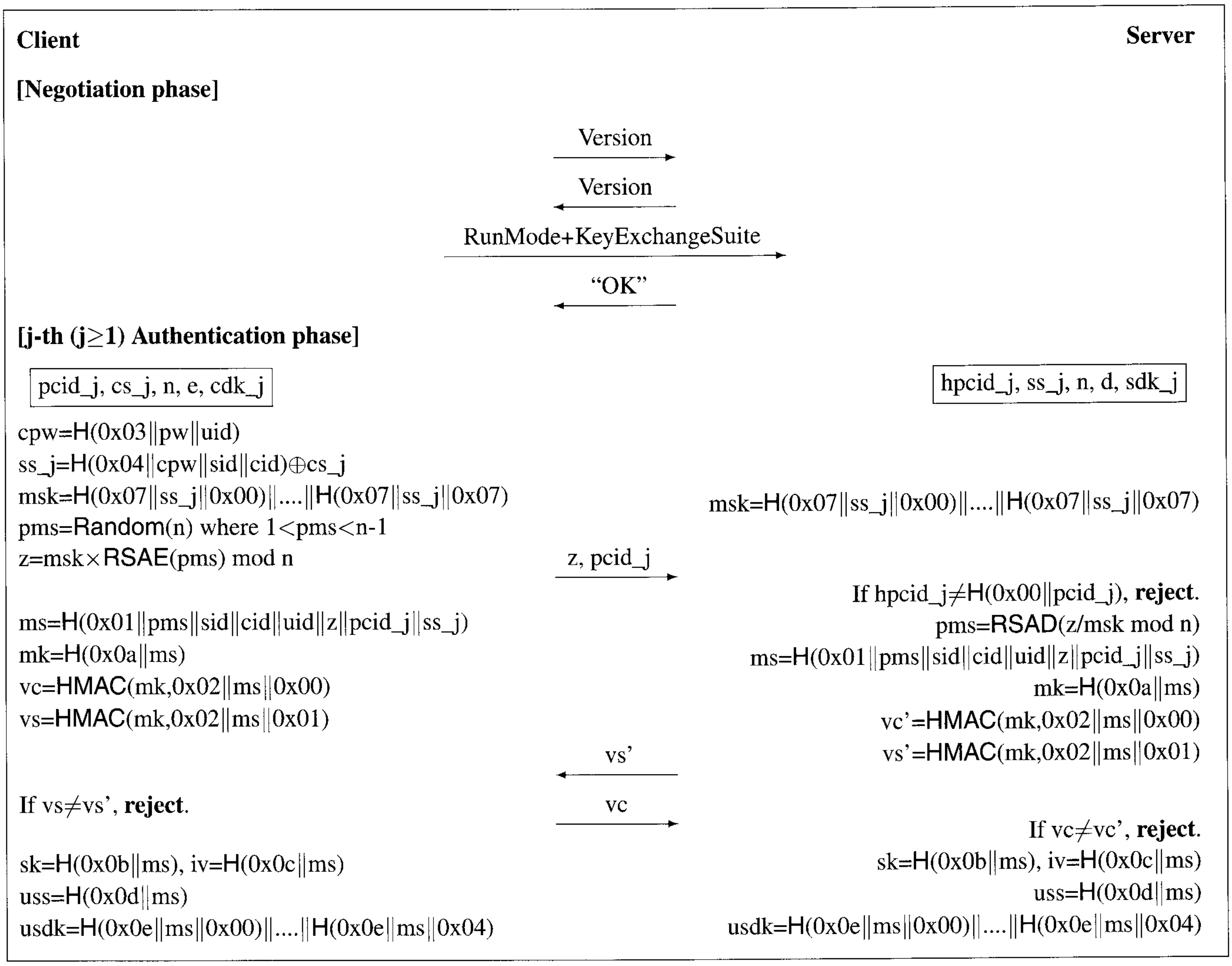


Fig. 6. SEC update protocol.

fact, this phase is crucial because without this phase an attacker can obtain significant information about the password [26]. The efficient parameters for "loop" can be determined by the result of [27]: loop=5 when e=257, loop=3 when e=12289, loop=2 when e=1179649, and loop=1 when e=274877906944l.[4] In the authentication phase, the client first encrypts a randomly-chosen pre-master secret pms with the RSA public key PK=(e,n) and then masks the ciphertext with msk, computed from the disposable password. The resultant value z is sent to the server. As the server knows the correct mask value and the RSA private key SK=(d,n), the pms and its derivative value (ms) are shared between the client and the server. After authenticating each other, they additionally generate a symmetric-key sk and an initial vector iv for message confidentiality, and a MAC key mk for data integrity. Note that these secrets can realize secure channels between the client and the server. Through the established secure channels, the client registers a one-time ID hpcid_1 to the server, and also performs the secret update (SEC update), the data key update (DK update) and/or the public key update (PK update) protocols.

---

[4] When e=3, loop=26.



Client **Server**

[Negotiation phase]

Client → Server: Version

Server → Client: Version

Client → Server: RunMode+KeyExchangeSuite

Server → Client: "OK"

[j-th ($j \geq 1$) Authentication phase]

Client: pcid_j, cs_j, n, e, cdk_j

Server: hpcid_j, ss_j, n, d, sdk_j

Client:
cpw=H(0x03||pw||uid)
ss_j=H(0x04||cpw||sid||cid)$\oplus$cs_j
msk=H(0x07||ss_j||0x00)||....||H(0x07||ss_j||0x07)
pms=Random(n) where 1<pms<n-1
z=msk$\times$RSAE(pms) mod n

Server:
msk=H(0x07||ss_j||0x00)||....||H(0x07||ss_j||0x07)

Client → Server: z, pcid_j

Server:
If hpcid_j$\neq$H(0x00||pcid_j), **reject.**
pms=RSAD(z/msk mod n)
ms=H(0x01||pms||sid||cid||uid||z||pcid_j||ss_j)
mk=H(0x0a||ms)
vc'=HMAC(mk,0x02||ms||0x00)
vs'=HMAC(mk,0x02||ms||0x01)

Client:
ms=H(0x01||pms||sid||cid||uid||z||pcid_j||ss_j)
mk=H(0x0a||ms)
vc=HMAC(mk,0x02||ms||0x00)
vs=HMAC(mk,0x02||ms||0x01)

Server → Client: vs'

Client:
If vs$\neq$vs', **reject.**

Client → Server: vc

Server:
If vc$\neq$vc', **reject.**

Client:
sk=H(0x0b||ms), iv=H(0x0c||ms)
uss=H(0x0d||ms)
usdk=H(0x0e||ms||0x00)||....||H(0x0e||ms||0x04)

Server:
sk=H(0x0b||ms), iv=H(0x0c||ms)
uss=H(0x0d||ms)
usdk=H(0x0e||ms||0x00)||....||H(0x0e||ms||0x04)

Fig. 7. Regular protocol (continue to Fig. 8) where the enclosed values in the rectangle represent stored secrets of client and server, respectively.

A.2 SEC Update Protocol

In the SEC update protocol, the user registers his/her password pw' to the server (see Fig. 6). In fact, the user registers a combined value ss_1 of a randomly-chosen number cs_1 and pw' to the server. At the end of this protocol, the client stores the secret cs_1 on memory and the server holds the authentication secret ss_1 on its database.

A.3 Regular Protocol

The regular protocol can be used for usual setup (j=0) and for the j-th ($j \geq 1$) protocol execution (see Figs. 7 and 8). Actually, this protocol is an extension of [19] in the sense that an authenticated client can recover the data key dk from his/her partial data key (cdk_j) and server's partial data key (sdk_j), transmitted through secure channels. That is, dk=cdk_j$\oplus$sdk_j$\oplus$dkmsk where dkmsk is a data-key mask computed from the password pw and uid. Note that the current stored secrets of client and server should be updated (overwritten) securely only if all the session transactions complete successfully.

In the negotiation phase, a pair of client and server determines which version of the protocol and which "RunMode+KeyExchangeSuite" would be used subsequently. If j=0, "RunMode+KeyExchangeSuite" is fixed to the setup of single mode. If $j \geq 1$, "RunMode+KeyExchangeSuite" is fixed to the regular protocol of single mode. At the start of the j-th ($j \geq 1$) authentication phase, the client stores j-th secrets (pcid_j,cs_j,n,e,cdk_j) on memory and the server holds the corresponding secrets (hpcid_j,ss_j,n,d,sdk_j) on its database. First, the client encrypts a randomly-chosen pre-master secret pms with the RSA public key PK=(e,n) and then masks the ciphertext with msk, computed from the password pw and the stored secret cs_j. The resultant value z and the one-time id pcid_j are sent to the server. As the server holds the authentication secret ss_j and the RSA private key SK=(d,n), the pms and its derivative value (ms) are shared between the client and the server. After authenticating each other, they additionally generate a symmetric-key sk and an initial vector iv for message confidentiality, a MAC key mk for data integrity, and update secrets (uss and usdk) for the current stored secrets. As we already explained before, the sk, iv and mk realize secure channels between the client and the server. Through the established secure channels, the server sends the current partial data key sdk_j to the client, who can easily retrieve

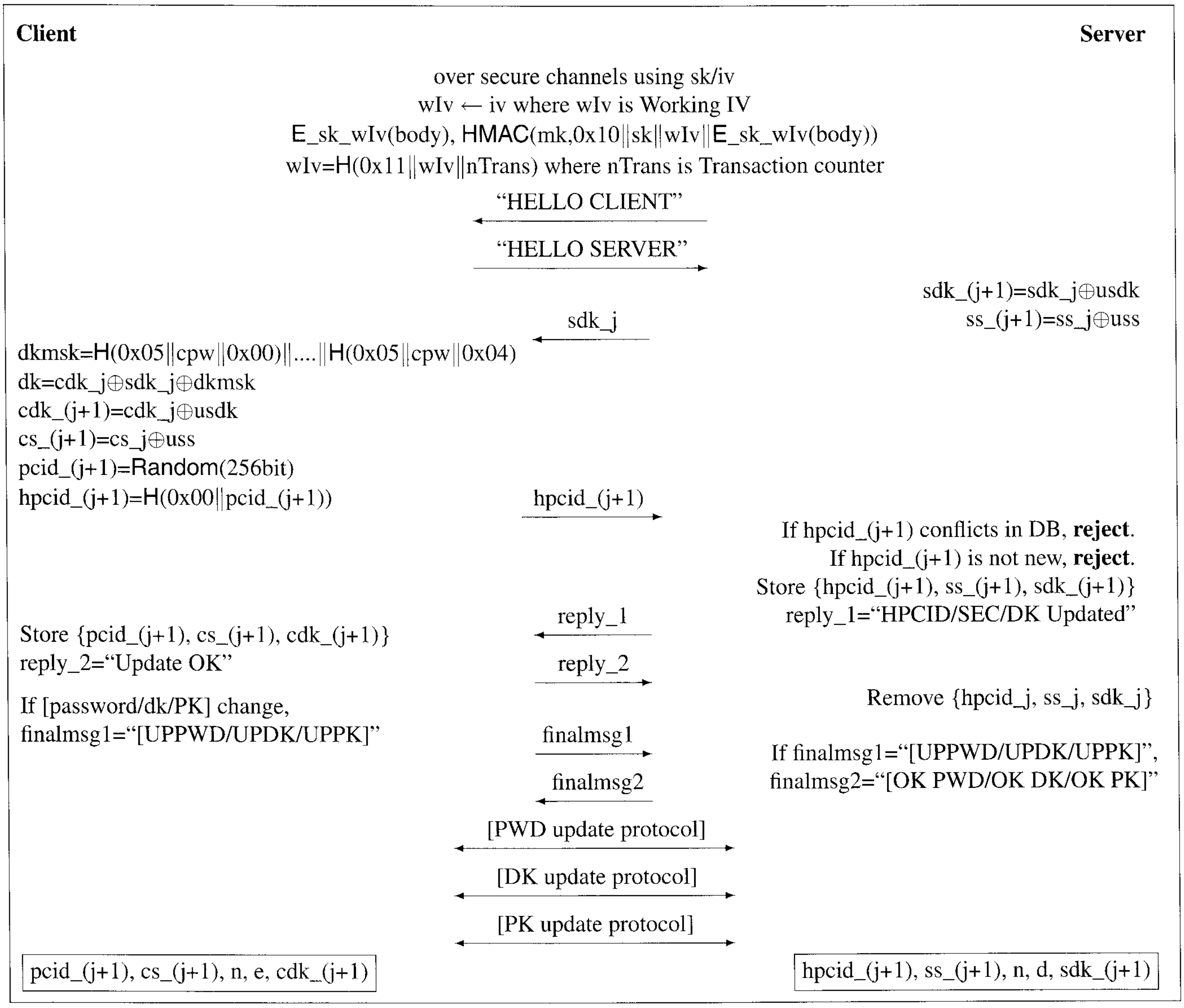
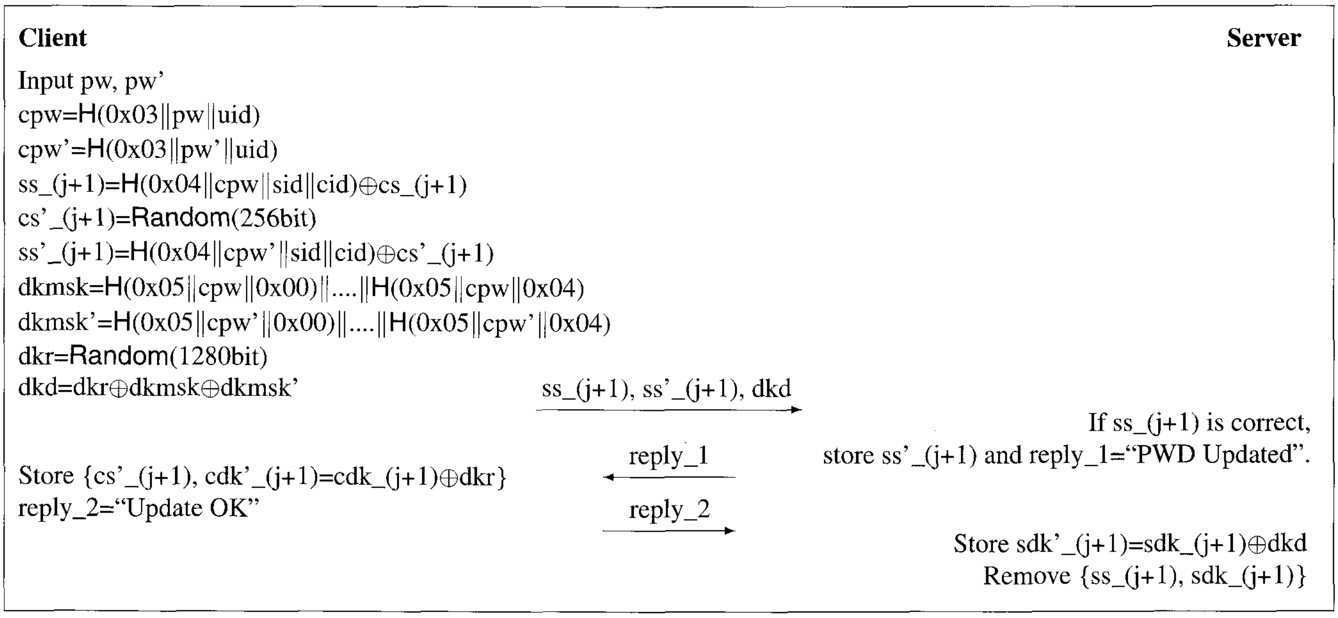


Fig. 8. Regular protocol (continue from Fig. 7) where the enclosed values in the rectangle represent stored secrets of client and server, respectively.

the data key dk with cdk_j⊕sdk_j⊕dkmsk where dkmsk is a data-key mask computed from the password pw and uid. At the same time, the client and the server update the current stored secrets ((cs_j,cdk_j) and (ss_j,sdk_j)) to new stored secrets ((cs_(j+1),cdk_(j+1)) and (ss_(j+1),sdk_(j+1))) with uss and usdk, respectively. In addition, the client registers a one-time ID hpcid_(j+1) to the server. If the client agrees with the server to update some information (i.e., password/data key/PK), the corresponding update protocols would be performed successively. If j=0, the PWD update and DK update protocols should be followed because the initial password is set to empty. At the end of the j-th (j≥1) authentication phase, the client stores (j+1)-th secrets (pcid_(j+1),cs_(j+1),n,e,cdk_(j+1)) on memory and the server holds the corresponding secrets (hpcid_(j+1),ss_(j+1),n,d,sdk_(j+1)) on its database for the next session. Note that, if the communications between the client and the server are disconnected in the updating process, either party should keep j-th and (j+1)-th stored secrets for the next authentication.

A.4 PWD Update Protocol

In the PWD update protocol, the user updates the password pw with a new password pw' (see Fig. 9). Recall that the password pw has been used for two different purposes: One is to generate the authentication secret ss_(j+1) and the other is to compute the data-key mask dkmsk. The client first computes ss_(j+1) from the password pw and cs_(j+1), and ss'_(j+1) from a new password pw' and a randomly-chosen number cs'_(j+1). After choosing a random number dkr, the client also computes dkd=dkr⊕dkmsk⊕dkmsk' where dkmsk (resp., dkmsk') is the data-key mask computed from the password pw (resp., pw'). Then, the client sends (ss_(j+1),ss'_(j+1),dkd) to the server. Finally, the client and the server update ((cs_(j+1),cdk_(j+1)) and (ss_(j+1),sdk_(j+1))) with ((cs'_(j+1),cdk'_(j+1)) and (ss'_(j+1),sdk'_(j+1))) where cdk'_(j+1)=cdk_(j+1)⊕dkr and sdk'_(j+1)=sdk_(j+1)⊕dkd. Of course, one can easily see that cdk'_(j+1)⊕sdk'_(j+1)⊕dkmsk'=cdk_(j+1)⊕dkr⊕sdk_(j+1) ⊕ dkd⊕dkmsk'=cdk_(j+1)⊕dkr⊕sdk_(j+1)⊕dkr⊕dkmsk⊕dkmsk' ⊕dkmsk'=cdk_(j+1)⊕sdk_(j+1)⊕dkmsk=cdk_(j+1)⊕sdk_(j+1)



Client

Input pw, pw'
cpw=H(0x03||pw||uid)
cpw'=H(0x03||pw'||uid)
ss_(j+1)=H(0x04||cpw||sid||cid)⊕cs_(j+1)
cs'_(j+1)=Random(256bit)
ss'_(j+1)=H(0x04||cpw'||sid||cid)⊕cs'_(j+1)
dkmsk=H(0x05||cpw||0x00)||....||H(0x05||cpw||0x04)
dkmsk'=H(0x05||cpw'||0x00)||....||H(0x05||cpw'||0x04)
dkr=Random(1280bit)
dkd=dkr⊕dkmsk⊕dkmsk'

ss_(j+1), ss'_(j+1), dkd →

Server

If ss_(j+1) is correct,
store ss'_(j+1) and reply_1="PWD Updated".

← reply_1

Client

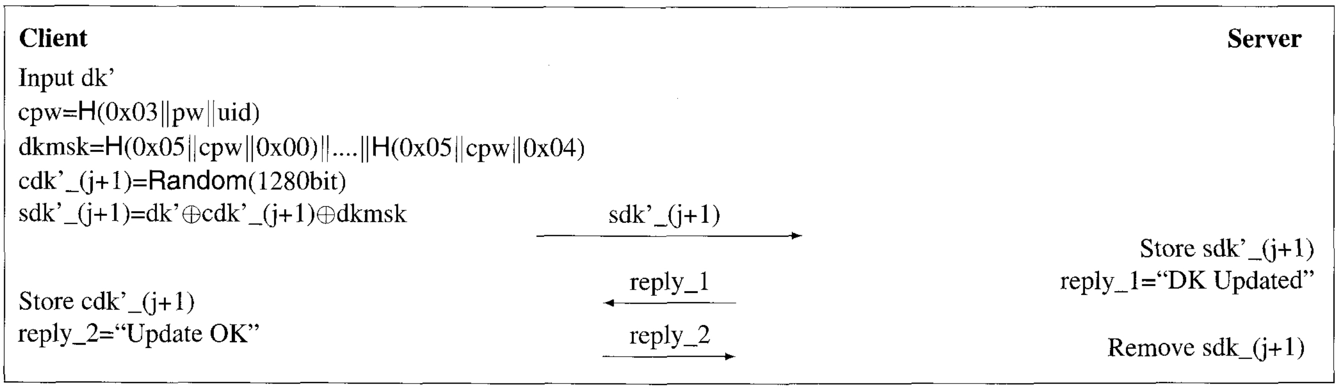
Store {cs'_(j+1), cdk'_(j+1)=cdk_(j+1)⊕dkr}
reply_2="Update OK"

reply_2 →

Server

Store sdk'_(j+1)=sdk_(j+1)⊕dkd
Remove {ss_(j+1), sdk_(j+1)}

Fig. 9. PWD update protocol where pw' is a new password.



Client

Input dk'
cpw=H(0x03||pw||uid)
dkmsk=H(0x05||cpw||0x00)||....||H(0x05||cpw||0x04)
cdk'_(j+1)=Random(1280bit)
sdk'_(j+1)=dk'⊕cdk'_(j+1)⊕dkmsk

sdk'_(j+1) →

Server

Store sdk'_(j+1)
reply_1="DK Updated"

← reply_1

Client

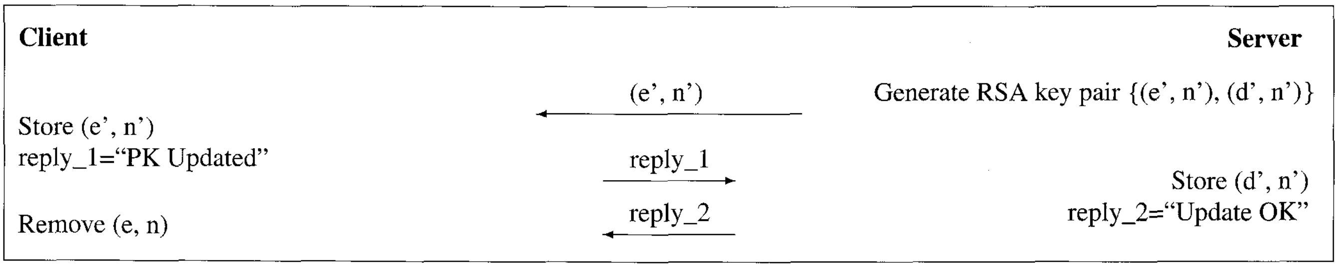
Store cdk'_(j+1)
reply_2="Update OK"

reply_2 →

Server

Remove sdk_(j+1)

Fig. 10. DK update protocol where dk' is a new 1280-bit data key.



Server

Generate RSA key pair {(e', n'), (d', n')}

← (e', n')

Client

Store (e', n')
reply_1="PK Updated"

reply_1 →

Server

Store (d', n')
reply_2="Update OK"

← reply_2

Client

Remove (e, n)

Fig. 11. PK update protocol where {(e',n'), (d',n')} is a new RSA key pair.

⊕dkmsk=dk.

A.5 DK Update Protocol

In the DK update protocol, the user updates the data key dk with a new 1280-bit data key dk' (see Fig. 10). The client chooses a random number cdk'_(j+1) and computes sdk'_(j+1)=dk'⊕cdk'_(j+1)⊕dkmsk where dkmsk is the data-key mask computed from the password pw. The sdk'_(j+1) is sent to the server. Finally, the client and the server update (cdk_(j+1) and sdk_(j+1)) with (cdk'_(j+1) and sdk'_(j+1)), respectively.

A.6 PK Update Protocol

In the PK update protocol, the server updates the RSA key pair (see Fig. 11). After generating a new RSA key pair (PK'=(e',n'),SK'=(d',n')), the server sends the PK' to the client. Finally, the client and the server update (PK and SK) with (PK' and SK'), respectively.

B. Discussions

This subsection summarizes security analysis and advantages of the proposed single mode for the leakage-resilient authentication and data management system.

B.1 Security Analysis

- **Security of data key:** The single mode of Section II-A provides a higher level of security for the data key dk. The data key dk remains information-theoretically secure even if either the stored secret (i.e., cdk_j) of client or the stored secret (i.e., sdk_j) of server is leaked out. It is clear that cdk_j and sdk_j can be viewed as shares of (2,2)-threshold secret sharing scheme [28]. Another security layer for the data key is that an attacker can not get any information about dk from cdk_i and sdk_j where $i \neq j$. Suppose that an attacker obtains cdk_j and sdk_(j+1). Since cdk_j⊕sdk_(j+1)=cdk_j⊕sdk_j⊕usdk=cdk_j⊕cdk_j⊕dk⊕dkmsk⊕usdk=dk⊕dkmsk⊕usdk, the data key dk is completely hidden with the update secret usdk. This also implies that the already-leaked secret (cdk_j or sdk_j) becomes obsolete if a pair of client and server successfully authenticates each other and update the current stored secrets. The final security layer for the data key is that, even if an attacker obtains cdk_j and sdk_j at the same time, the attacker has to do off-line dictionary attacks on the password pw in order to retrieve dk.
- **Security of password:** The single mode provides almost same level of security for the password pw as that for the data key dk. The password pw is information-theoretically secure, even if either the stored secret (i.e., cs_j) of client or the stored secret (i.e., ss_j) of server is leaked out, because of the same reason as above. Also, an attacker can not get any information about pw from cs_i and ss_j where $i \neq j$. Suppose that an attacker obtains cs_(j+1) and ss_j. Since cs_(j+1)⊕ss_j=cs_j⊕uss⊕H(0x04||cpw||sid||cid)⊕cs_j=uss⊕H(0x04||cpw||sid||cid), the password remains secure with the update secret uss. Of course, automatic revocation functionality of leaked secrets (cs_j or ss_j) is valid as well. If an attacker obtains all the stored secrets of client and impersonates the client, only serial on-line dictionary attacks are possible where the attacker tests a password candidate one by one until the client and the server successfully authenticate each other. As a final security layer for the password, an attacker who obtains cs_j and ss_j at the same time should perform off-line dictionary attacks on the password pw.
- **Security of session key:** As in [19], the security of session key sk can be guaranteed against an attacker who obtains either all the stored secrets of client or the RSA private key SK of server (refer to Theorem 1 and 2 of [19]). Of course, the attacker can do any kinds of active attacks (e.g., eavesdropping, messages modification, impersonation, man-in-the-middle attacks). Unfortunately, if an attacker obtains the authentication secret ss_j, the attacker can freely impersonate the client after intercepting the first message (z,pcid_j) from the legitimate client.
- **"Strong" forward secrecy:** The single mode provides forward secrecy in the sense that exposure of the long-term secrets does not compromise security of the previously-established session keys (refer to Theorem 4 of [19]). Moreover, "strong" forward secrecy can be guaranteed because the previous communications remain "private", as long as the leakage of stored secrets did not happen, even in the case that the underlying public-key encryption (i.e., RSA) or its computational problem is completely broken or solved. Suppose that an attacker obtains all the (j+1)-th stored secrets of client and server and is trying to compute the j-th session key sk. The goal of the attacker is to compute the pre-master secret pms from z with all the available secrets. However, the attacker can not compute the sk because dividing ss_(j+1) into ss_j and uss is impossible without knowing the pms.

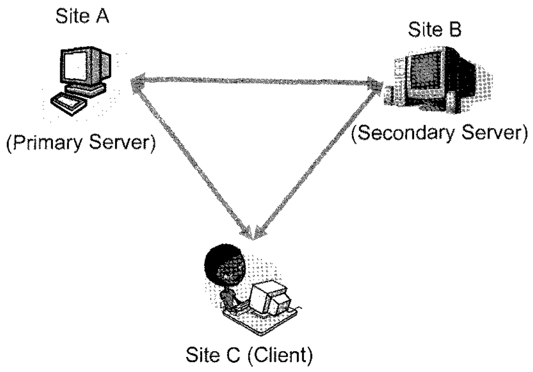


Fig. 12. Cluster mode.

B.2 Advantages

- **Easy-of-use:** Even though a client communicates with many different servers, a user remembers only one (short) password. Instead, the number of secrets (stored on client) grows linearly with the number of servers.
- **Simple management:** The management of the single mode is simple because it does not need any public-key certificate and CRL (Certificate Revocation Lists). In addition, this single mode is resistant to Phishing attacks since the user always need to use the password pw and the stored secret cs_j.
- **Automatic revocation of leaked secrets:** As we already explained above, this functionality is preferable in restricting the number of on-line dictionary attacks on the password pw.
- **Computational efficiency on client side:** The j-th protocol execution of the single mode is extremely efficient in terms of computational cost, required for client, since the RSA public key encryption can be used with the public exponent e=3 and thus the total computational cost is just 3 modular multiplications. Also, note that the RSA encryption with e=3 (i.e., 2 modular multiplications) is pre-computable. Therefore, this single mode can be applied to computationally-restricted mobile phones/devices or PDA.

III. CLUSTER MODE

A. Each Protocol for Cluster Mode

In Section II, we proposed the single mode for the leakage-resilient authentication and data management system and showed that it can provide a higher level of security for the data key dk. Though the single mode is applicable to any kind of two-party setting, a potential problem is that the user can not retrieve the data key dk when one of the parties is unavailable or physically broken/destroyed. In order to provide availability of the data key, we introduce a cluster mode, for the leakage-resilient authentication and data management system, which allows a user to recover dk securely even in the case of one party's compromise. In the cluster mode, there are three parties (site A, B, and C) where site A plays a role of server for both site B and C, site B plays a role of server for site C and a role of client for site A, and site C plays a role of client for both site A and B (see Fig. 12). Irrationally, we denote site A and B by primary server and secondary server, respectively.

As the basic structure is the same as one in the single mode, we omit the overall transition flow for the cluster mode. The main difference is that each site has to store two different stored secrets. For the data key dk, site A stores (sdk_ba,sdk_ca), site B stores (cdk_ba,sdk_cb) and site C stores (cdk_ca,cdk_cb) so that the user can retrieve dk from any pair of sites: dk⊕dkmsk=cdk_ca⊕sdk_ca=cdk_cb⊕sdk_cb=cdk_ba⊕sdk_ba. However, the cluster mode would be more complicated than the single mode because we have to maintain synchronization among the three parties (i.e., site A, B, and C). Remind that a higher level of security for the data key can be achieved by updating and synchronizing stored secrets between the client and the server in the single mode.

From here on, we briefly explain each sub-protocol for the cluster mode.

A.1 Cluster Initialization Protocol

The cluster initialization protocol is designed for easy setup where a user just remembers disposable passwords (pw_ca and pw_cb) and the corresponding servers store hashed values (hpw_ca and hpw_cb) of each password (see Fig. 13). As in the initialization protocol, these disposable passwords are only valid for a short period of time in order to avoid Denial-of-Service (DoS) attacks.

First, site A and C perform the initialization protocol with hpw_ca and pw_ca, and then they can generate the stored secrets of site A and C. Next, site B and C also perform the initialization protocol with hpw_cb and pw_cb, and then they can generate the stored secrets of site B and C. Now, the remaining works of the cluster initialization protocol is to generate the stored secrets of site A and B, and distribute them securely. Of course, these works should be done through secure channels, established between site A and C and between site B and C.

After choosing a random number pcid_ba1 and computing a one-time ID hpcid_1, site C registers hpcid_ba1 to site A and pcid_ba1 to site B. As in the SEC update protocol, site C chooses a random number cs_ba1 and computes the corresponding authentication secret ss_ba1 from the password pw and cs_ba1. Then, the values ss_ba1 and cs_ba1 are registered to site A and B, respectively. As in the DK update protocol, site C chooses a random number cdk_ba1 and derives sdk_ba1=dk⊕cdk_ba1⊕dkmsk where dkmsk is the data-key mask computed from the password pw. Then, the values sdk_ba1 and cdk_ba1 are registered to site A and B, respectively. Finally, site C completes the cluster initialization protocol by registering site A's public key PK_a1 to site B as in the PK update protocol. At the end of this protocol, site A and B hold the stored secrets (hpcid_ba1,ss_ba1,sdk_ba1,SK_a1) and (pcid_ba1,cs_ba1,cdk_ba1,PK_a1), respectively.

A.2 Cluster Setup Protocol

The cluster setup protocol is used for off-line setup where site C is off-line but it can be manually setup by site B (see Fig. 14).

First, site A and B perform the regular protocol with the respective stored secrets, and then they can update the current stored secrets with new ones and also realize secure channels between site A and B. The subsequent message exchanges between site A and B should be done securely with the established secure channels. After generating two pairs of setup parameter, site B registers the stored secrets (hpcid_ca1,ss_ca1,sdk_ca1) to site A and holds the stored secrets (hpcid_cb1,ss_cb1,sdk_cb1) on its own database. Finally, site B registers two stored secrets (pcid_ca1,cs_ca1,cdk_ca1,PK_a1) and (pcid_cb1,cs_cb1,cdk_cb1,PK_b1) to site C off-line.

A.3 Cluster Regular Protocol

The cluster regular protocol can be used for usual setup (j=0) and for the j-th (j≥1) protocol execution (see Fig. 15).

First, site A and C perform the regular protocol with the respective stored secrets, and then they can update the current stored secrets with new ones, and also realize secure channels between site A and C. Next, site B and C also perform the regular protocol with the respective stored secrets, and then they can update the current stored secrets with new ones, and realize secure channels between site B and C. The remaining works of the cluster regular protocol is to generate the stored secrets of site A and B, and distribute them securely. Of course, these works should be done through secure channels, established between site A and C and between site B and C.

After choosing a random number pcid_ba(j+1), usdk_ba and uss_ba, site C computes a one-time ID hpcid_(j+1) and registers (hpcid_ba(j+1),usdk_ba,uss_ba) to site A and (pcid_ba(j+1),usdk_ba,uss_ba) to site B. With the received information (hpcid_ba(j+1),usdk_ba,uss_ba), site A updates (hpcid_baj,ss_baj,sdk_baj) with (hpcid_ba(j+1),ss_ba(j+1),sdk_ba(j+1)) where sdk_ba(j+1)=sdk_baj⊕usdk_ba and ss_ba(j+1)= ss_baj⊕uss_ba. In the same way, site B updates (pcid_baj,cs_baj, cdk_baj) with (pcid_ba(j+1),cs_ba(j+1),cdk_ba(j+1)) where cdk_ba(j+1) =cdk_baj⊕usdk_ba and cs_ba(j+1)=cs_baj⊕uss_ba. If site A, B, and C agree to update some information (i.e., password/data key/PK), the corresponding update protocols would be performed successively. At the end of this protocol, site A and B hold the stored secrets (hpcid_ba(j+1),ss_ba(j+1),sdk_ba(j+1),SK_a(j+1)) and (pcid_ba(j+1),cs_ba(j+1),cdk_ba(j+1),PK_a(j+1)), respectively.

If j=0, the cluster PWD update and cluster DK update protocols should be followed because the initial password is set to empty. As in the regular protocol, if the communications among site A, B and C are disconnected in the updating process, either party should keep j-th and (j+1)-th stored secrets for the next authentication.

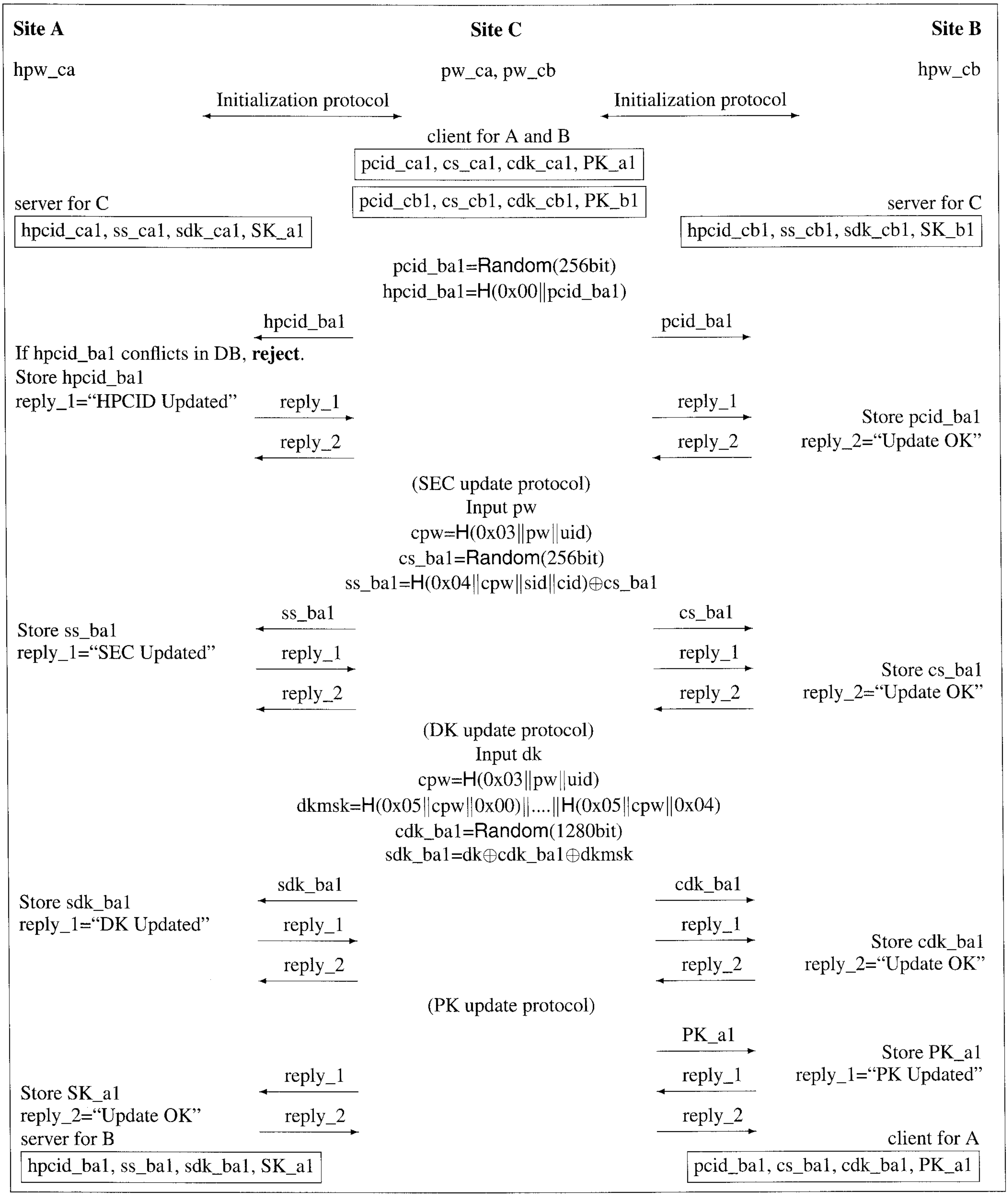


Fig. 13. Cluster initialization protocol where the enclosed values in the rectangle represent stored secrets of site A, B, and C, respectively.

A.4 Cluster PWD Update Protocol

In the cluster PWD update protocol, the user updates the password pw with a new password pw' among site A, B, and C (see Fig. 16). If only one of the servers (site A or B) is available, the cluster PWD update protocol should not be performed (otherwise, synchronization of the password pw and the data key dk is broken).

First, site A and C perform the cluster regular and PWD update protocols with the respective stored secrets, and then they

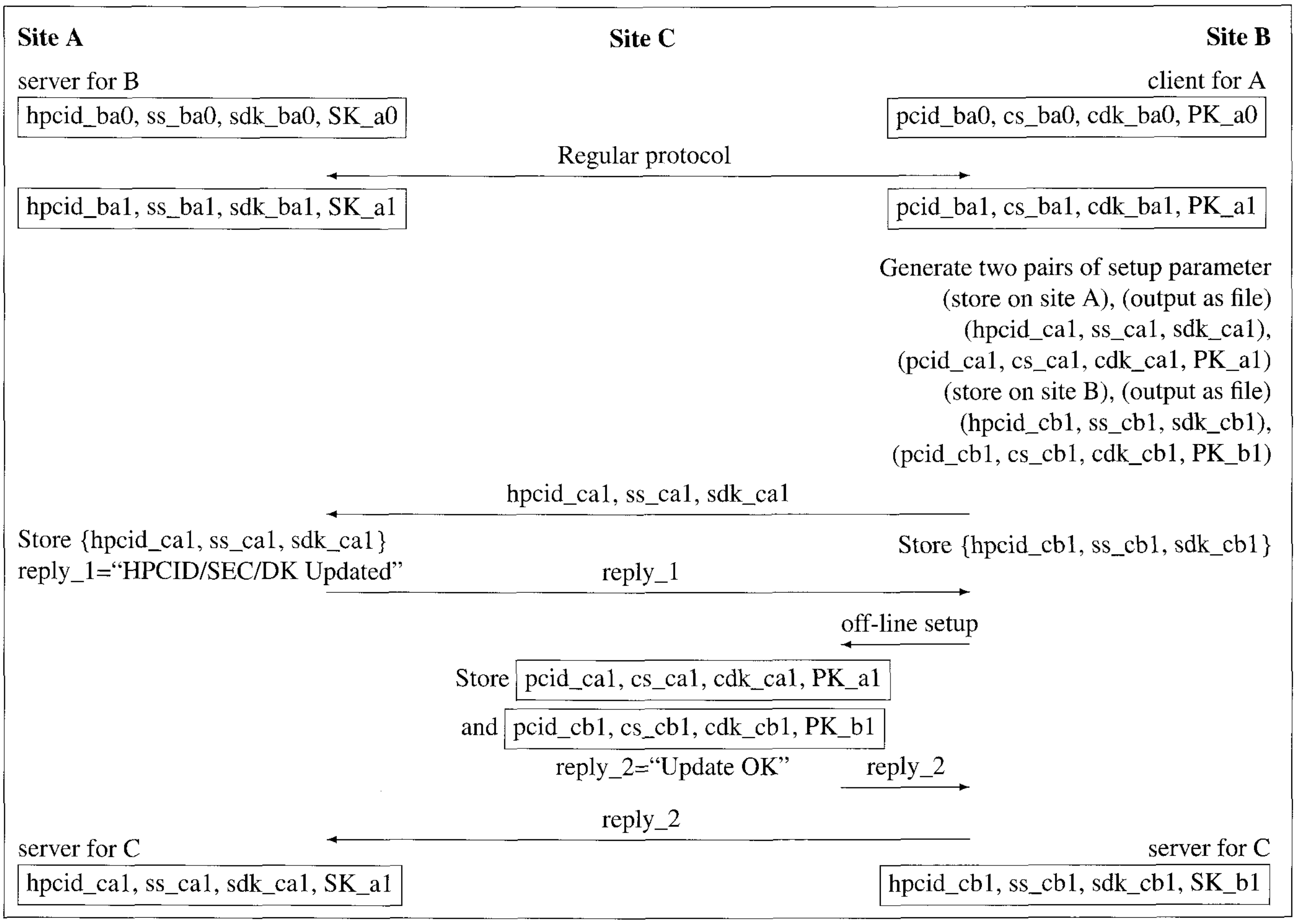


Fig. 14. Cluster setup protocol where the enclosed values in the rectangle represent stored secrets of site A, B, and C, respectively.

can update the current stored secrets with new ones and also realize secure channels between site A and C. Next, site B and C also perform the cluster regular and PWD update protocols with the respective stored secrets, and then they can update the current stored secrets with new ones and realize secure channels between site B and C. Because of the cluster regular protocol, site A and B can update the current stored secrets with new ones as well. Now, the remaining works of the cluster PWD update protocol is to update the password pw with a new one pw' between site A and B securely. Of course, these works should be done through secure channels, established between site A and C and between site B and C.

Recall that the password pw has been used for the authentication secret ss_ba(j+1) and the data-key mask dkmsk. The site C first computes ss_ba(j+1) from the password pw and cs_ba(j+1), and ss'_ba(j+1) from a new password pw' and a randomly-chosen number cs'_ba(j+1). After choosing a random number dkr, site C also computes dkd=dkr$\oplus$dkmsk$\oplus$dkmsk' where dkmsk (resp., dkmsk') is the data-key mask computed from the password pw (resp., pw'). Then, site C sends (ss_ba(j+1),ss'_ba(j+1),dkd) to site A, and sends (cs_ba(j+1),cs'_ba(j+1),dkr) to site B. Finally, site A and B update (ss_ba(j+1),sdk_ba(j+1)) and (cs_ba(j+1),cdk_ba(j+1)) with (ss'_ba(j+1),sdk'_ba(j+1)) and (cs'_ba(j+1),cdk'_ba(j+1)), respectively, where cdk'_ba(j+1)= cdk_ba(j+1)$\oplus$dkr and sdk'_ba(j+1)=sdk_ba(j+1)$\oplus$dkd. Of course, one can easily see that cdk'_(j+1)$\oplus$sdk'_(j+1)$\oplus$dkmsk' =dk. Note that dkr is used for randomizing dkmsk and dkmsk'.

A.5 Cluster DK Update Protocol

In the cluster DK update protocol, the user updates the data key dk with a new 1280-bit data key dk' among site A, B, and C (see Fig. 17). If only one of the servers (site A or B) is available, the cluster DK update protocol should not be performed (otherwise, synchronization of the password pw and the data key dk is broken).

First, site A and C perform the cluster regular and DK update protocols with the respective stored secrets, and then they can update the current stored secrets with new ones and also realize secure channels between site A and C. Next, site B and C also perform the cluster regular and DK update protocols with the respective stored secrets, and then they can update the current stored secrets with new ones and realize secure channels between site B and C. Because of the cluster regular protocol, site A and B can update the current stored secrets with new ones as well. Now, the remaining works of the cluster DK update protocol is to update the data key dk with a new one dk' between site A and B securely. Of course, these works should be done through secure channels, established between site A and C and between site B and C.

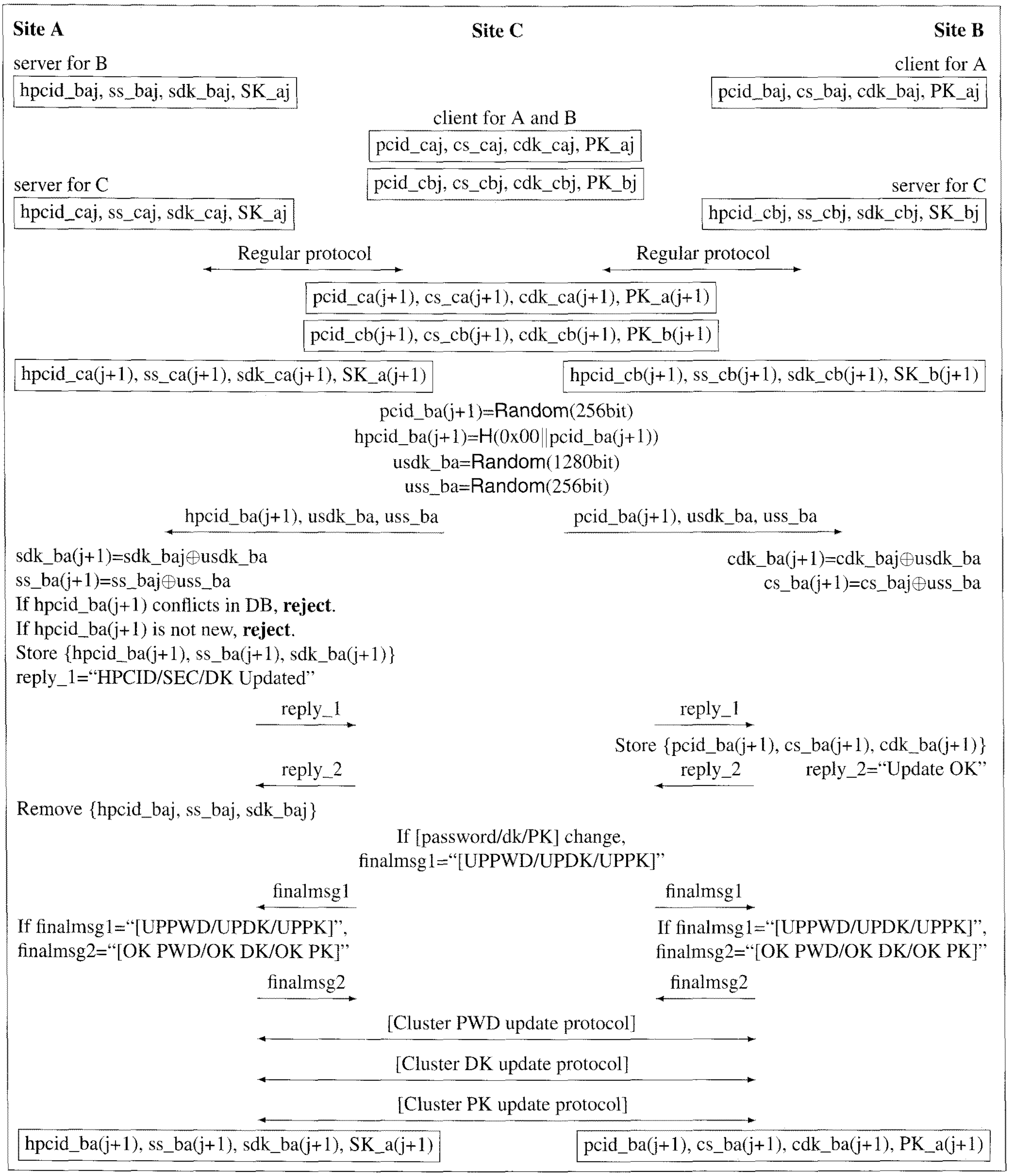


Fig. 15. Cluster regular protocol where the enclosed values in the rectangle represent stored secrets of site A, B, and C, respectively.

The site C chooses a random number cdk'_ba(j+1) and computes sdk'_ba(j+1)=dk'⊕cdk'_ba(j+1)⊕dkmsk where dkmsk is the data-key mask computed from the password pw. Then, site C sends sdk'_ba(j+1) to site A, and sends cdk'_ba(j+1) to site B. Finally, site A and B update (sdk_ba(j+1) and cdk_ba(j+1)) with (sdk'_ba(j+1) and cdk'_ba(j+1)), respectively.

A.6 Cluster PK Update Protocol

In the cluster PK update protocol, the primary and secondary servers (site A and B) update their RSA public keys (PK_a(j+1) and PK_b(j+1)) with new ones (PK'_a(j+1) and PK'_b(j+1)) among site A, B, and C (see Fig. 18).

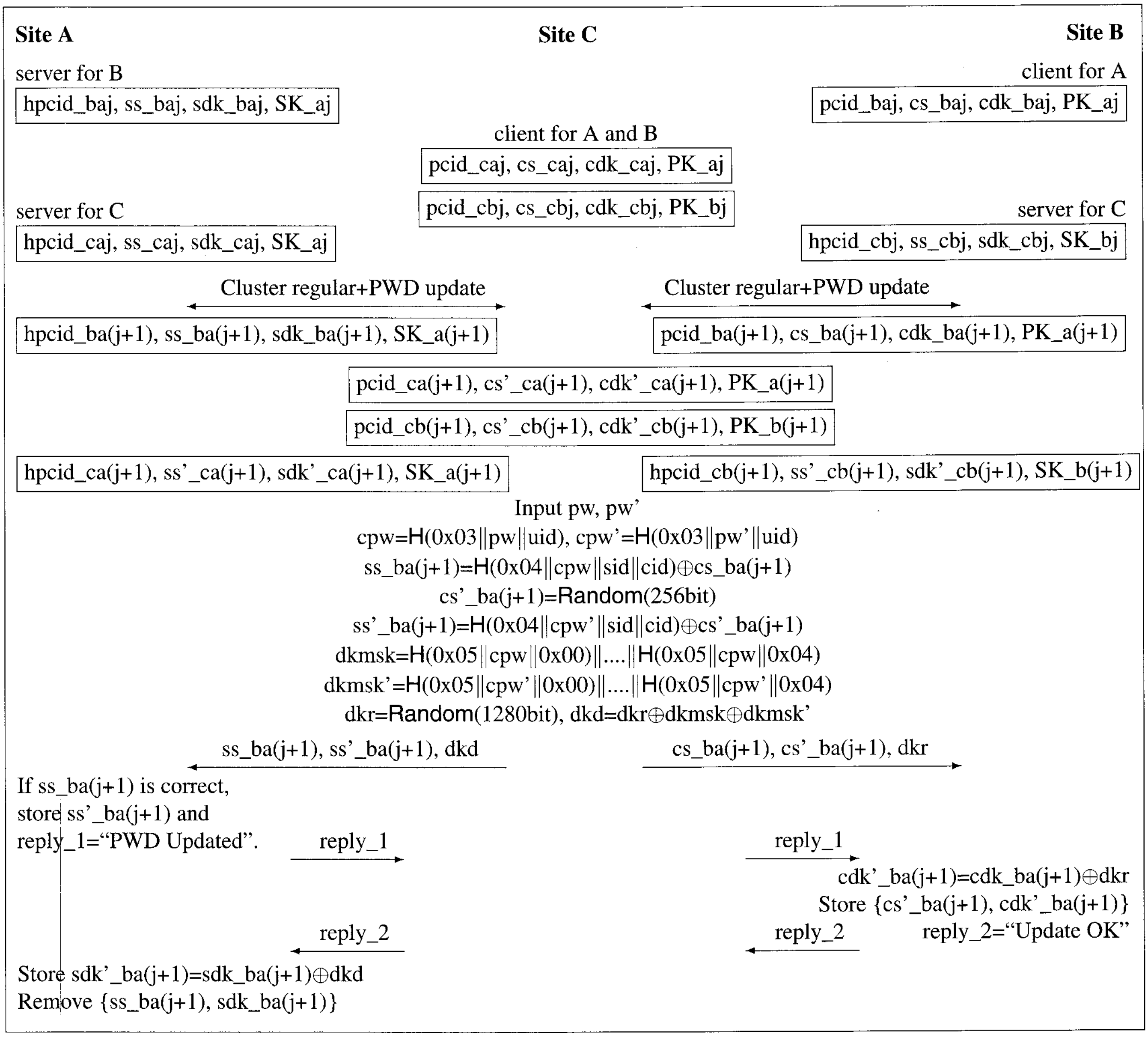


Fig. 16. Cluster PWD update protocol where pw' is a new password and the enclosed values in the rectangle represent stored secrets of site A, B, and C, respectively.

First, site A and C perform the cluster regular and PK update protocols with the respective stored secrets, and then they can update the current stored secrets with new ones and also realize secure channels between site A and C. Next, site B and C also perform the cluster regular and PK update protocols with the respective stored secrets, and then they can update the current stored secrets with new ones and realize secure channels between site B and C. Because of the cluster regular protocol, site A and B can update the current stored secrets with new ones as well. Now, the remaining works of the cluster PK update protocol is to update the RSA public key PK_a(j+1) with a new one PK'_a(j+1) between site A and B securely. Of course, these works should be done through secure channels, established between site A and C and between site B and C. Actually, site C just sends PK'_a(j+1) to site B. Finally, site A and B update (SK_a(j+1) and PK_a(j+1)) with (SK'_a(j+1) and PK'_a(j+1)), respectively.

B. Discussions

In this subsection, we discuss availability of the data key and several security analysis in the cluster mode for the leakage-resilient authentication and data management system.

B.1 Availability of Data Key

- **In collapse of site A:** In the regular protocol, the user mutually authenticates with the secondary server (i.e., site B) by using the client (i.e., site C) and then recovers the data key dk as follows: dk=cdk_cbj⊕sdk_cbj⊕dkmsk. That is, on-line data key retrieval is possible.
- **In collapse of site B:** It is similar with the above case. In the regular protocol, the user mutually authenticates with the primary server (i.e., site A) by using the client

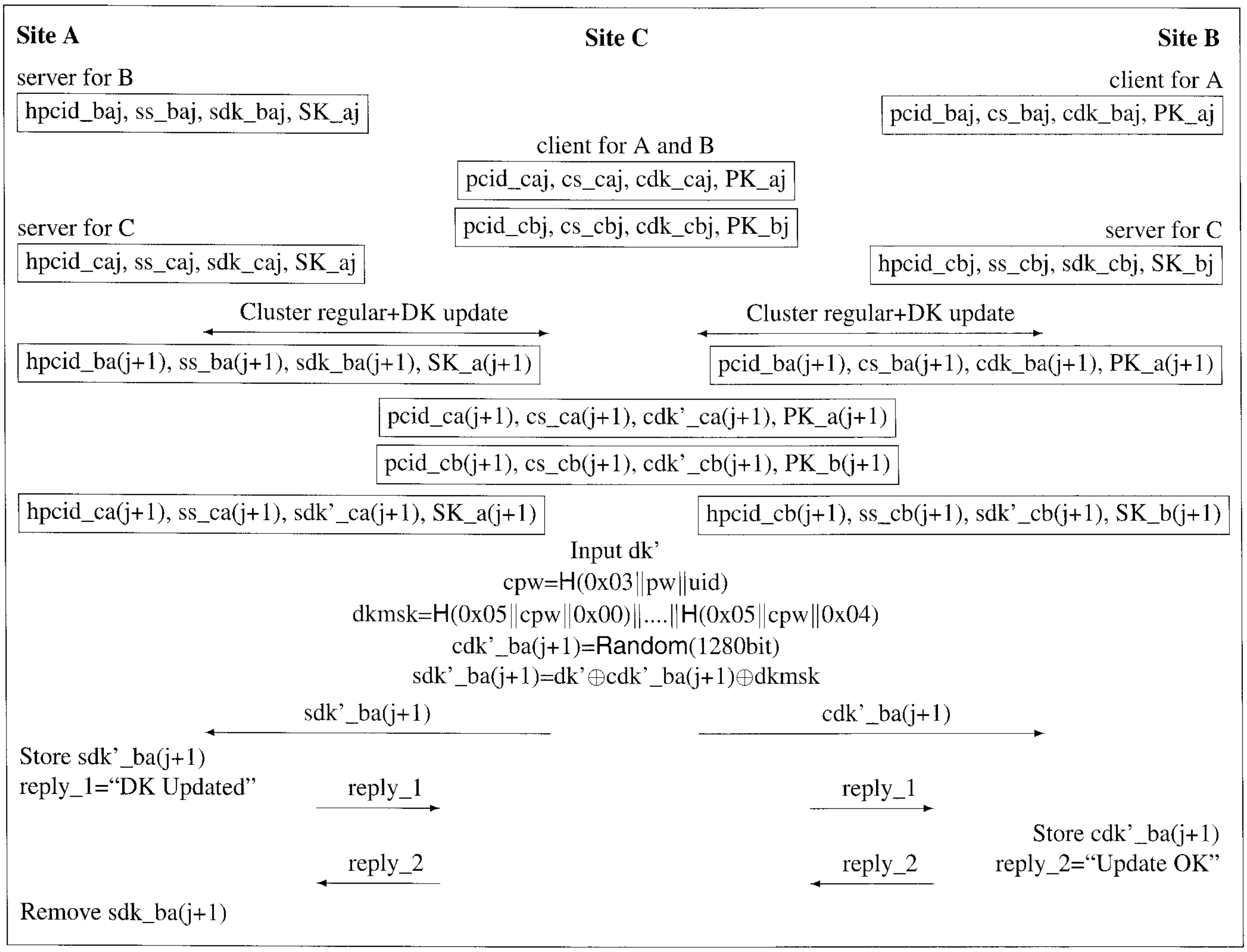


Fig. 17. Cluster DK update protocol where dk' is a new 1280-bit data key and the enclosed values in the rectangle represent stored secrets of site A, B, and C, respectively.

(i.e., site C) and then recovers the data key dk as follows: dk=cdk_caj⊕sdk_caj⊕dkmsk. That is, on-line data key retrieval is possible.

- **In collapse of site C:** In order to recover the data key, the user should go to the secondary server (i.e., site B) and perform the regular protocol with the primary server (i.e., site A). If authentication is successful, the user recovers the data key dk as follows: dk=cdk_baj⊕sdk_baj⊕dkmsk.

B.2 Security Analysis

- **Security of data key:** In the cluster mode of Section III-A, the data key dk is distributed among site A, B and C such that any pair of parties can recover dk. If the stored secret of one site (A, B, or C) is leaked out, the data key dk remains information-theoretically secure. Suppose that an attacker obtains the stored secret (i.e., sdk_baj and sdk_caj) of site A. Since sdk_baj=dk⊕cdk_baj⊕dkmsk and sdk_caj=dk⊕cdk_caj⊕dkmsk, the attacker can not get any information about the data key dk. As in the single mode, an attacker also can not get any clue on dk from (cdk_cai,cdk_cbi), (cdk_baj,sdk_cbj), and (sdk_cak,sdk_bak) where i≠j≠k. That means, the stored secret of each site is leaked out in a different time slot. Suppose that an attacker obtains (cdk_ca(j-1),cdk_cb(j-1)), (cdk_baj,sdk_cbj), and (sdk_ca(j+1),sdk_ba(j+1)). One can easily see that the data key dk is completely hidden with the update secret usdk. This also implies that the already-leaked secret (cdk_baj,sdk_cbj) becomes obsolete if site C (or A) and site B successfully authenticate each other and update the current stored secrets with new ones (i.e., automatic revocation of leaked secrets). The final security layer for the data key dk is that, even if an attacker obtains two stored secrets from any two sites at the same time, the attacker has to do off-line dictionary attacks on the password pw in order to retrieve dk.
- **Security of password:** The cluster mode provides almost same level of security for the password pw as that for the data key dk. The password pw is information-theoretically secure, even if the stored secret of one site (A, B, or C) is leaked out, because of the same reason as above. Also, an attacker can not get any information about pw from the stored secrets (cs_cai,cs_cbi), (cs_baj,ss_cbj), and (ss_cak,ss_bak), leaked in a different time slot, where i≠j≠k. Suppose that an attacker obtains (cs_ca(j-1),cs_cb(j-1)) of site C,

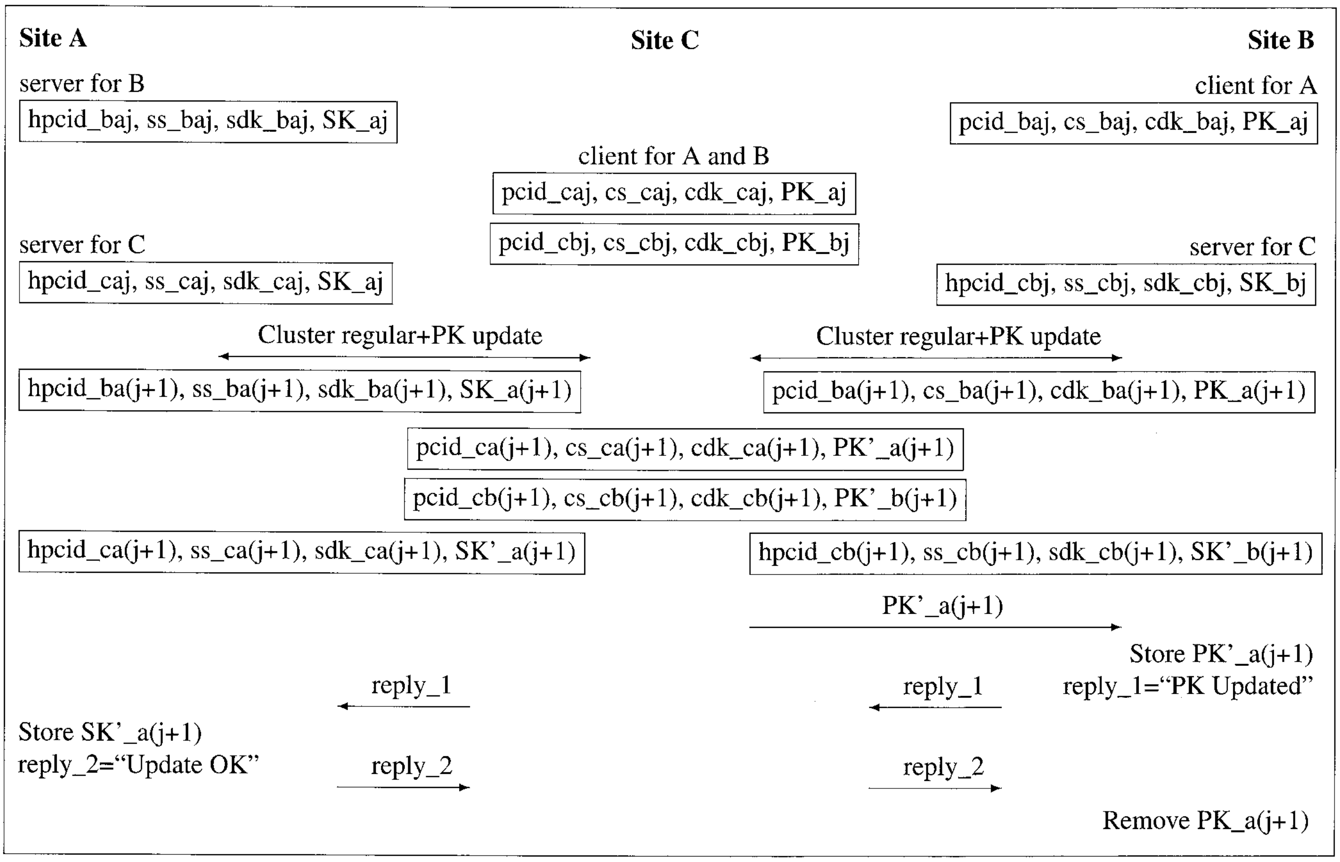


Fig. 18. Cluster PK update protocol where (PK'_a(j+1), SK'_a(j+1)) is a new RSA key pair and the enclosed values in the rectangle represent stored secrets of site A, B, and C, respectively.

(cs_baj,ss_cbj) of site B and (ss_ca(j+1),ss_ba(j+1)) of site A. It is clear that the password pw remains secure with the update secret uss. However, the attacker can do serial on-line dictionary attacks by impersonating site C and site B with the leaked secrets (cs_ca(j-1),cs_cb(j-1)) and (cs_baj,ss_cbj), respectively. As we already explained before, these on-line dictionary attacks are possible until any pair of parties perform the cluster regular protocol successfully (i.e., automatic revocation of leaked secrets). Without any leaked secrets, an attacker can not do even serial on-line dictionary attacks. As a final security layer for the password, an attacker who obtains two stored secrets from any two sites at the same time should perform off-line dictionary attacks on the password pw.

- **Security of session key:** The security of session key in the single mode can be easily extended to the cluster mode. Even though all the stored secrets of site C are leaked out, the security of session key sk is guaranteed since the success probability of on-line dictionary attacks is small. If an attacker obtains the RSA private keys SK_aj (of site A) and SK_bj (of site B) at the same time, the session key sk is also secure because the authentication among three sites totally depends on a high-entropy secret ss. If an attacker obtains the authentication secrets (ss_baj and ss_caj) of site A, the attacker can impersonate site B and C after intercepting the first message pcid_baj from site B and pcid_caj from site C, respectively. If an attacker obtains the authentication secrets (cs_baj and ss_cbj) of site B, the attacker can impersonate only site C after intercepting the first message pcid_cbj from site C. Note that in the last two cases the security of session key is not guaranteed, however, the attacker can not recover the data key dk from the leaked secrets.
- **"Strong" forward secrecy:** The cluster mode also provides forward secrecy in the sense that exposure of the long-term secrets does not compromise security of the previously-established session keys. Moreover, "strong" forward secrecy can be guaranteed because the previous communications remain "private," as long as the leakage of stored secrets from any site did not happen, even in the case that the underlying public-key encryption (i.e., RSA) or its computational problem is completely broken or solved.

IV. CONCLUSIONS

In this paper, we first clarified the problems for network storage and showed two requirements of the data key (i.e., a higher level of security and availability). In order to achieve a higher level of security for the data key, we have proposed the single mode that is a natural extension of the leakage-resilient authentication protocol, and discussed its security analysis and advantages. For availability of the data key, we have proposed the cluster mode (based on the single mode) where the key is distributed among three parties so that any pair of legitimate parties can recover the data key at any time. Though the cluster mode is more complicated than the single mode, its security is comparable to that of the single mode so that both modes for the leakage-resilient authentication and data management system provide a maximum level of security against active attacks as well as leakage of stored secrets from any parties. Finally, we also stress that our proposed system can work with any previous specific storage technologies for data confidentiality and data integrity since it is a solution to the data-key protection and availability.

REFERENCES

[1] Amazon, "Amazon simple storage service (Amazon S3)." [Online]. Available: http://aws.amazon.com/s3

[2] BitTorrent Inc., "BitTorrent." [Online]. Available: http://www.bittorrent.com

[3] E. L. Miller, W. E. Freeman, D. D. E. Long, and B. C. Reed, "Strong security for network-attached storage," in *Proc. USENIX Conference on File and Storage Technologies*, Jan. 2002.

[4] E. J. Goh, H. Shacham, N. Modadugu, and D. Boneh, "SiRiUS: Securing remote untrusted storage," in *Proc. Network and Distributed System Security*, 2003, pp. 131–145.

[5] E. Mykletun, M. Narasimha, and G. Tsudik, "Authentication and integrity in outsourced databases," *ACM Trans. Storage*, vol. 2, no. 2, pp. 107–138, 2006.

[6] A. Heizmann, B. Palazzi, C. Papamanthou, and R. Tamassia, "Efficient integrity checking of untrusted network storage," in *Proc. 4th ACM International Workshop on Storage Security and Survivability*, 2008, pp. 43–54.

[7] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. Eurocrypt2003*, 2003, LNCS 2656, pp. 416–432.

[8] M. T. Goodrich, R. Tamassia, and A. Schwerin, "Implementation of an authenticated dictioinary with skip lists and commutative hashing," in *Proc. DARPA Information Survivability Conference and Exposition II*, 2001, pp. 68–82.

[9] IETF, "PPP extensible authentication protocol (EAP)," RFC 2284, March 1998.

[10] B. Aboba, L. Blunk, J. Vollbrecht, J. Carlson, and H. Levkowetx, "Extensible authentication protocol (EAP)," IETF RFC 3748, June 2004.

[11] A. Frier, P. Karlton, and P. Kocher, "The SSL 3.0 protocol," Netscape Communication Corp. [Online]. Available: http://wp.netscape.com/eng/ssl3

[12] IETF, "Transport layer security (tls) charter." [Online]. Available: http://www.ietf.org/html.charters/tls-charter.html

[13] D. Harkins and D. Carrel, "The Internet Key Exchange (IKE)," IETF RFC 2409, Nov. 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2409.txt

[14] C. Kaufman, "Internet key exchange (IKEv2) protocol," IETF RFC 4306, Dec. 2005.

[15] IEEE P1363, "IEEE standard specifications for public key cryptography," Nov. 1999.

[16] IEEE P1363.2, "Standard specifications for password-based public key cryptographic techniques." [Online]. Available: http://grouper.ieee.org/groups/1363/passwdPK/submissions.html

[17] S. H. Shin, K. Kobara, and H. Imai, "Leakage-resilient authenticated key establishment protocols," in *Proc. Asiacrypt2003*, 2003, LNCS 2894, pp. 155–172.

[18] S. H. Shin, K. Kobara, and H. Imai, "A simple leakage-resilient authenticated key establishment protocol, its extensions, and applications," *IEICE Trans. Fund. Electronics, Commun. and Computer Sciences*, vol. E88-A, no. 3, pp. 736–754, Mar. 2005.

[19] S. H. Shin, K. Kobara, and H. Imai, "An efficient and leakage-resilient RSA-based authenticated key exchange protocol with tight security reduction," *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, vol. E90-A, no. 2, pp. 474-490, 2007.

[20] R. Richardson, "CSI survey 2007: The 12th annual computer crime and security survey," Computer Security Institute, http://www.gocsi.com/forms/csi_survey.jhtml, 2007.

[21] Federal information processing standards publication 180-2, "Secure hash standard (SHS)," Aug. 2002. [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-2/fips180-2withchangenotice.pdf

[22] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash fucntions for message authentication," in *Proc. Crypto'96*, 1996, LNCS 1109, pp. 1–15.

[23] R. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, 1978.

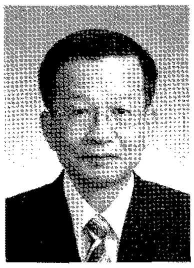
[24] M. Bellare and P. Rogaway, "Random oracles are practical: A paradigm for designing efficient protocols," in *Proc. ACM CCS'93*, 1993, pp. 62–73.

[25] M. Bellare and P. Rogaway, "The exact security of digital signatures: How to sign with RSA and Rablin," in *Proc. Eurocrypt'96*, 1996, LNCS 1070, pp. 399–416.

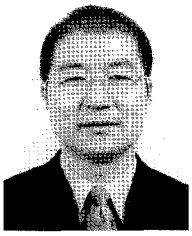
[26] S. Patel, "Number theoretic attacks on secure password schemes," in *Proc. IEEE Symposium on Security and Privacy*, 1997, pp. 236–247.

[27] S. H. Shin, K. Kobara, and H. Imai, "RSA-based password-authenticated key exchange, revisited," *IEICE Trans. Inf. Syst.*, vol. E91-D, no. 5, pp. 1424–1438, 2008.

[28] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.



Hideki Imai was born in Shimane, Japan on May 31, 1943. He received the B.E., M.E., and Ph.D. degrees in electrical engineering from the University of Tokyo in 1966, 1968, and 1971, respectively. From 1971 to 1992 he was on the faculty of Yokohama National University. From 1992 to 2006 he was a Professor in the Institute of Industrial Science, the University of Tokyo. In 2006 he was appointed as an Emeritus Professor of the University of Tokyo and a Professor of Chuo University. Concurrently he serves as the Director of Research Center for Information Security, National Institute of Advanced Industrial Science and Technology. His current research interests include information theory, coding theory, cryptography, and information security. From IEICE (the Institute of Electronics, Information and Communication Engineers), He received Best Book Awards in 1976 and 1991, Best Paper Awards in 1992, 2003, and 2004, Yonezawa Memorial Paper Award in 1992, Achievement Award in 1995, Inose Award in 2003, and Distinguished Achievement and Contributions Award in 2004. He also received Golden Jubilee Paper Award from the IEEE Information Theory Society in 1998, Wilkes Award from the British Computer Society in 2007, Official Commendations from the Minster of Internal Affairs and Communications in June 2002 and from the Minister of Economy, Trade and Industry in October 2002. He was awarded Honor Doctor Degree by Soonchunhyang University, Korea in 1999 and Docteur Honoris Causa by the University of Toulon Var, France in 2002. He is also the recipient of the Ericsson Telecommunications Award 2005 and the Okawa Prize 2008. He is a member of the Science Council of Japan. He was elected a Fellow of IEEE, IEICE and IACR (International Association for Cryptologic Research) in 1992, 2001, and 2007, respectively. He is an IEEE Life Fellow since 2009. He has chaired many committees of scientific societies and organized a number of international conferences. He served as the President of the Society of Information Theory and its Applications in 1997, of the IEICE Engineering Sciences Society in 1998, and of the IEEE Information Theory Society in 2004. He is currently the Chair of CRYPTREC (Cryptography Techniques Research and Evaluation Committee of Japan).



SeongHan Shin received the 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, the University of Tokyo. From April 2007 to March 2008, he was a visiting researcher in the Institute of Science and Engineering, Chuo University, Tokyo, Japan. From April 2006, he has been working for the Research Center for Information Security, National Institute of Industrial Science and Technology, Japan, as a research scientist of the Research Team for Security Fundamentals. From October 2008, he also has been serving as an associate professor in the Center for Research and Development Initiative, Chuo University, Japan. He received the CSS Student Paper Award and the IWS2005/WPMC2005 Best Student Paper Award in 2003 and 2005, respectively. His research interests include information security, cryptography and wireless security.

Kazukuni Kobara received his B.E. degree in electrical engineering and M.E. degree in computer science and system engineering from the Yamaguchi University in 1992, 1994, respectively. He also received his Ph.D. degree in engineering from the University of Tokyo in 2003. From 1994 to 2000 and 2000 to 2006 he was a technical associate and a research associate respectively for the Institute of Industrial Science of the University of Tokyo. In 2006, he joined the National Institute of Advanced Industrial Science and Technology (AIST) where he was the leader of the Research Team for Security Fundamentals in the Research Center for Information Security (RCIS). Currently he is a chief research scientist at RCIS. His research interests include cryptography, information and network security. He received the SCIS Paper Award and the Vigentennial Award from ISEC group of IEICE in 1996 and 2003, respectively. He also received the Best Paper Award of WISA, the ISITA Paper Award for Young Researchers, the IEICE Best Paper Award (Inose Award), the WPMC Best Paper Award and the JSSM Best Paper Award in 2001, 2002, 2003, 2005, and 2006, respectively. He is a Member of IEICE and IACR. He served as a Member of CRYPTREC (2000–present), the vice chairperson of WLAN security committee (2003) and the chief investigator of INSTAC identity management committer (2007–present).