PAPER *Special Section on Formal Approaches*

Verification of Group Key Management of IEEE 802.21 Using ProVerif*

Ryoga NOGUCHI[†a)], ***Nonmember***, **Yoshikazu HANATANI**[††b)], ***and*** **Kazuki YONEYAMA**[†c)], ***Members***

SUMMARY Home Energy Management Systems (HEMS) contain devices of multiple manufacturers. Also, a large number of groups of devices must be managed according to several clustering situations. Hence, since it is necessary to establish a common secret group key among group members, the group key management scheme of IEEE 802.21 is used. However, no security verification result by formal methods is known. In this paper, we give the first formal verification result of secrecy and authenticity of the group key management scheme of IEEE 802.21 against insider and outsider attacks using ProVerif, which is an automatic verification tool for cryptographic protocols. As a result, we clarify that a spoofing attack by an insider and a replay attack by an outsider are found for the basic scheme, but these attacks can be prevented by using the scheme with the digital signature option.

key words: *ProVerif, IEEE 802.21, HEMS, group key management*

1. Introduction

In recent years, Home Energy Management Systems (HEMS) have been popularized, and various devices that compose HEMS are connected to the HEMS network. For example, in Japan, ECHONET Lite [5] standardized by IEC 62394 [6] and ISO/IEC 14543-4-3 [7] is recommended as a standard interface for HEMS. Also, various application protocols for HEMS are developed. HEMS consists of devices of multiple manufacturers and it is necessary to manage a large number of groups efficiently. Therefore, the group key management (GKM) scheme of IEEE 802.21 [2] is used to establish a common group key for each group. Although some existing works [3], [27] gave security analysis of the scheme, hand-written security proofs are error-prone. In addition, these security proofs are the verifications in all options are valid, and the security of each option is not considered. Hence, security verification by formal methods is important in order to guarantee rigorous security. However, no verification result of the GKM scheme of IEEE 802.21 by using formal methods is known. In this paper, we give the first formal verification result of secrecy and authenticity of the GKM scheme of IEEE 802.21 against insider and outsider attacks using ProVerif [4] which is an automatic verification tool for cryptographic protocols. Since the GKM scheme of IEEE 802.21 has two modes (one is the basic mode without the digital signature option, and the other is the mode with the digital signature option.), we examine security of both modes.

ProVerif guarantees security in the Dolev-Yao model [24] if no attack is found by assuming underlying cryptographic primitives are ideally secure. ProVerif can verify various security requirements such as secrecy and authenticity. We note that ProVerif does not support verification of general multi-party protocols in which the number of parties is not fixed. Therefore, we formalize a representative setting of the GKM scheme of IEEE 802.21 (i.e., including a group manager (GM), two group members and a non-group member). Naor et al. [25] proposed the Complete Subtree (CS) method, and proved that ciphertexts of all qualified sets of group members in the CS are secure against non-group members, and the GKM scheme of IEEE 802.21 are fully managed by the CS method. Hence, if our representative setting is secure, then it is expected that other settings for qualified sets of group members are also secure.

We verify several levels of secrecy and authenticity of the GKM scheme in IEEE 802.21, and Table 1 summarizes our verification result. First, even a device such that insider but non-group member cannot obtain any information of the group key (i.e., strong secrecy is guaranteed). On the other hand, without the digital signature option, insiders can perform a spoofing attack that impersonates the GM who distributes the group key, and outsiders can perform a replay attack. Also, if the digital signature option is adopted, these attacks are not found. From these results, it is clarified that the digital signature option actually improves the security.

There is a technical subtlety with the verification of the GKM scheme with the digital signature option against replay attacks because GM and devices must manage incremented sequence numbers. It means that each party must keep the current sequence number as a state. However, ProVerif itself has no functionality to keep states. We firstly examine an idea to formalize this situation by using the Table functionality of ProVerif. The table in ProVerif keeps contents over sessions. However, the output of ProVerif verifying injective correspondence is "*cannot be proved*" because ProVerif cannot judge whether the same event will be executed again after a content of the table is updated. Hence, we show a verification against replay attacks by formalizing

Manuscript received November 9, 2020.
Manuscript revised March 15, 2021.
Manuscript publicized July 14, 2021.
†The authors are with Ibaraki University, Hitachi-shi, 316–8511 Japan.
††The author is with Toshiba Corporation, Kawasaki-shi, 212–8581 Japan.
*This paper is the full version of the extended abstract appeared in [1].
a) E-mail: 19nm725f@vc.ibaraki.ac.jp
b) E-mail: yoshikazu.hanatani@toshiba.co.jp
c) E-mail: kazuki.yoneyama.sec@vc.ibaraki.ac.jp
DOI: 10.1587/transinf.2020FOP0003

a concrete situation of replay attacks and verifying reachability of such an event.

1.1 Comparison to Previous Work

Our verification result differs from security analyses in [3], [27] as follows: First, the security analysis in [3] is for a device authentication protocol based on the GKM scheme in IEEE 802.21. However, the security proof of the part of the GKM scheme is omitted. Next, in [27], indistinguishability of the group key of the GKM scheme (with the digital signature option) is proved. However, security of the GKM scheme without the digital signature option is not analysed. Indistinguishability capture strong secrecy (i.e., the adversary cannot obtain any information about the group key of the target session) and authenticity in a combined way. The adversary is allowed to reveal several secret information such as long-term device keys, randomness in a session, and established group keys of sessions other than the target session. Also, the adversary can observe a polynomial-size number of sessions. On the other hand, we verify strong secrecy and authenticity of two modes (with-/without the digital signature option) of the GKM scheme for insider and outsider attacks. In particular, we show attacks to the GKM scheme without the digital signature option, that is not analysed in [27]. Also, a merit of using ProVerif is that verified security properties are easy to understand (for non-cryptographer) because each property can be verified in a separated way while the cryptographic security model in [27] is the combination of multiple properties and complicated. Outsider attacks correspond to leakage of long-term device keys. However, leakage of randomness and past group keys is not covered in our verification. Since unfixed number of processes cannot be handled due to the restriction of ProVerif, our verification explicitly fixes the number of sessions observed by the adversary.

1.2 Related Work

There have been several studies about security verification of standardized cryptographic protocols with formal methods. ProVerif and CryptoVerif [18] have been used to verify other standardized protocols such as ZRTP [19], SNMPv3 [20], mutual authentication protocol for GSM [21], LTE [22], OpenID Connect protocol [23], and others. Cremers and Horvat [8] verified 30 key exchange protocols standardized in ISO/IEC 11770-2 [9] and ISO/IEC 11770-3 [10] with automated security verification tool Scyther [11], and found some spoofing attacks. Scyther and Tamarin Prover [12] have been also used to verify other standardized protocols such as IKEv1/v2 in IPsec [13], entity authentication protocols in ISO/IEC 9798 [14], TLS 1.3 [15], 5G-AKA [16] and smart grid authentication protocol SAv5 in IEEE 1815-2012 [17] and others. Dreier et al. [26] showed verification results of industrial protocols OPC-UA and MODBUS with Tamarin Prover, and gave a modeling of sequence numbers and counters. While their analysis focuses on flow integrity, our analysis focuses on confidentiality (secrecy and strong secrecy) and authenticity that imply flow integrity.

2. ProVerif

ProVerif idealizes messages, communication channel, bitstrings, and others to symbols. Thus, security of cryptographic protocols can be verified in the Dolev-Yao model [24]. The verification method is based on the applied pi-calculus and Horn clauses. ProVerif can prove that an attack process (trace) violating a security requirement exists or not. If no trace can be derived, no attack against the target protocol exists. If a trace can be derived, an attack against the target protocol exists and ProVerif outputs a concrete attack procedure.

2.1 Secrecy

ProVerif can verify the following two levels of secrecy.

- Secrecy
  It can be verified whether an attacker can obtain the whole secret. It corresponds to one-way security in cryptography.
- Strong secrecy
  It can be verified whether an attacker can distinguish a process that the secret is replaced with a different symbol from the original process. It corresponds to indistinguishability in cryptography.

2.2 Authenticity

ProVerif can verify the following two kinds of authenticity.

- Non-injective correspondence
  It can be verified whether the communicating peer is actually the intended party or not. When an event is executed, ProVerif verifies that the corresponding event was executed in advance for any process. If non-injective correspondence is false, a spoofing attack exists.
- Injective correspondence
  Similar to non-injective correspondence, when an event is executed, ProVerif verifies that the corresponding event was executed only once in advance for any process. If non-injective correspondence is true but injective correspondence is false, a replay attack exists.

3. Group Key Management of IEEE 802.21

3.1 Overview

In the GKM scheme of IEEE 802.21, group keys are managed by using a method called the Group Key Block method by applying the Complete Subtree (CS) method [25].

Table 1 Verification result

	outsider attack			insider attack		
w/o signature option	secrecy	strong secrecy	authenticity	secrecy	strong secrecy	authenticity
	secure	secure	replay attack	secure	secure	spoofing attack
	outsider attack			insider attack		
w/ signature option	secrecy	strong secrecy	authenticity	secrecy	strong secrecy	authenticity
	secure	secure	secure	secure	secure	secure

"secure" means that no attack is found. "X attack" means that an attack is found, which corresponds to attack category X.

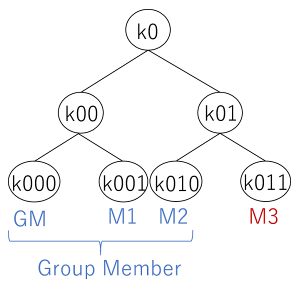


Fig. 1 Management Tree of Device Keys

Table 2 Device Keys

GM	k0, k00, k000
M1	k0, k00, k001
M2	k0, k01, k010
M3	k0, k01, k011

Specifically, the devices participating in the group share device keys in advance and manage them with the complete binary tree structure (called the management tree) as shown in Fig. 1. Each device key is assigned to nodes in the management tree and is indexed, GM and each device correspond to leaf nodes, and have device keys of nodes in the route to the root node as shown in Table 2.

When GM distributes a new group key to a group, GM randomly generates a group key, finds a subtree containing only leaf nodes corresponding to the devices belonging to the group (called CS), and encrypts the group key with the device keys corresponding to the root node of the subtree (in the case of Fig. 1, due to select a CS containing only GM, M1 and M2, k00 and k010 are used because GM and M1 can decrypt the ciphertext encrypted with k00, and M2 can decrypt the ciphertext encrypted with k010, but M3 cannot decyrpt any ciphertext with M3's device keys (k0,k01,k011)) and distribute ciphertexts by multicast. The devices contained in the group can obtain the group key by decrypting the received ciphertext with one of the device keys. When revoking a device from the group, GM executes a new session of the distribution protocol with the CS consisting of only members excluding the revoked device. (i.e., IEEE 802.21 specifies no dedicated protocol for revoking.) For example, if M1 is revoked from the CS containing (GM,M1,M2), GM randomly generates a new group key, and sends ciphertexts of the group key encrypted with k000 and k010. Hence, the revoked device (M1) cannot know the new group key because M1 does not have any device key to decrypt ciphertexts.

By this method, it is possible to reduce the computational and communication costs at the key distribution, as compared with the case of encrypting the group key by individual device keys and distributing by unicast. The disadvantage of this method is that devices must keep device keys in addition to the group key.

IEEE 802.21 has the digital signature option. In the basic mode (i.e., without the option), GM sends ciphertexts with other session information (including header information, the source ID, the destination IDs, the group ID, the transport address, and CS). In the mode with the digital signature option, GM also manages a sequence number and the number is incremented when the session is complete. The message from GM contains the sequence number and the signature of the message is also sent. Then, it is expected that spoofing attacks and replay attacks are prevented. However, there is no formal verification result.

3.2 Protocol

We recall the protocol of the GKM scheme. KDF is a key derivation function which is publicly shared. Wrap is a key wrapping algorithm and Unwrap is a key unwrapping algorithm using an authenticated encryption scheme. '(DS option)' in the beginning of a step means that the step is only contained in the mode with the digital signature option where *Gen* is a key generation, $Sign$ is a signature generation and $Verify$ is a signature verification algorithm. Although IEEE 802.21 internally divides GM and each user into MISUser and MISF, it is omitted here for simplicity because communications between MISUser and MISF are done inside a machine; and thus, messages between them are not revealed nor interrupted. Also, we consider the case that communications among GM and users are public (i.e., corresponding to using broadcast group *BG* as all destination groups *DG* in IEEE 802.21).

(1) Setup(n, M):

1. Let 2^n be the number of (potential) users managed by the group manager GM, and let devices M be a set of all users. GM generates a key tree T with depth n, and assigns (I_i, k_i) to each node in T where I_i is a node index represented as a binary string of length between 1 to n and k_i is a randomly generated node key where i corresponding to the node index I_i.
2. (DS option) For digital signature, GM generates public/secret key (pk,sk) with *Gen*.

3. For each user Mi in M, GM assigns Mi to a leaf node in T. Let $Path_{Mi}$ be a set of node indices of nodes from the leaf node which is assigned to Mi along the path to the root node. GM assigns $DK_i = (I_j, k_j)_{I_j \in Path_{M_i}}$, to Mi as the long-term keys.
4. GM sends DK_i to each of Mi via a secure channel.
5. (DS option) Additionally GM sends pk to each of M_i via a secure channel.
6. GM maintains list $GroupMasterInfo$ containing (group ID, group members, sequence number, security association identifier, master group key) for each session, and stores the initial value $(BG, M, 0, Null, Null)$.
7. Mi maintains list $GroupMembershipInfo_i$ containing (group ID, sequence number, security association identifier, master group key) for each session, and stores the initial value $(BG, 0, Null, Null)$.

(2) GM($T, \{DK_i\}, GroupMasterInfo, S$):

1. Let S be a set of group members which is a target for group key distribution.
2. Randomly generate a group identifier GI which identifies a group using the distributed group key for S.
3. Pick a sequence number SN for GI.
4. Randomly generate a master group key **mgk** $\in \{0, 1\}^l$ and select a security association identifier **SAID** which is an identifier of a group session key gsk = KDF(**mgk**).
5. Compute a list of indices CS from $M \backslash S$ and T by the CS method.
6. For all $I_i \in CS$, compute c_i = Wrap(k_i, **mgk**) where (I_i,k_i) is a node of T, and adds c_i to a group key data $GKD = GKD||c_i$.
7. (DS option) Pick a sequence number sq for the destination group.
8. (DS option) Compute $sig = Sign(sk, GI||SN||CS||GKD||$ **SAID**$||sq)$.
9. Send ($GI||SN||CS||GKD||$**SAID**) to users.
10. (DS option) Additionally send $(sq||sig)$ to users.
11. Store ($GI, S, SN,$ **SAID**, **mgk**) to $GroupMasterInfo$.

(3) Receiver Mi($DK_i, GroupMembershipInfo_i$):

1. Receive ($GI||SN||CS||GKD||$**SAID**).
2. (DS option) Additionally receive $(sq||sig)$.
3. (DS option) Check sq whether the received message is not a replay attack. If the message with sq was already accepted, Mi halts.
4. (DS option) If the verification of sig with $Verify$ is *false*, Mi halts.
5. If Mi has $(I_j, k_j) \in DK_i$ such that $I_j \in CS$, compute **mgk** = Unwrap(k_j, c_k) where $c_k \in GKD$ is the ciphertext corresponding with I_j.
6. Store ($GI, SN,$ **SAID**, **mgk**) to $GroupMembershipInfo_i$.

3.3 Security Requirements

The GKM scheme requires several security properties. First, the shared group key **mgk** must be hidden for entities except devices participating the session. It corresponds to secrecy and strong secrecy in ProVerif. Strong secrecy is the stronger notion than secrecy. Next, it must be prevented to impersonate GM to devices. It corresponds to authenticity in ProVerif. In this paper, we consider two situations about authenticity. One is spoofing attacks that the adversary corrupting devices not participating the session tries to impersonate GM to devices in the session. The other is replay attacks that devices accept multiple sessions between GM even if GM executes a session with devices only once. Spoofing attacks are captured by non-injective correspondence, and replay attacks are captures by injective correspondence.

4. Formalization of Protocol without Digital Signature Option

In this section, we formalize the basic mode (without the digital signature option) of the GKM scheme executed among GM, M1 and M2 in Fig. 1 (i.e., S =(M1,M2)). We examine the insider attack by M3 and the outsider attack because we must fix the number of parties for verification by ProVerif.

We verify secrecy, strong secrecy, and authenticity for non-injective correspondence and injective correspondence. Authenticity for non-injective correspondence captures security against spoofing attacks (i.e., the adversary tries to impersonate a device in the target group.). Authenticity for injective correspondence captures security against replay attacks (i.e., the adversary tries to make GM accept two messages if just one of these has already been sent by the device.). The formalization of these properties is shown in Sect. 4.5.

4.1 Declaration and Function Definition

```
free c:channel. (* public channel for BG *)
free cgm:channel [private]. (* private
    channel for GM *)

free mgk:bitstring [private]. (* master
    group key *)

type DeviceKey.
type GroupIdentifier.
type SequenceNumber.
type CompleteSubtree.
type SecurityAssociationIdentifier.

type node.
free n0, n00, n01, n000, n001, n010, n011:
    node.

(* key (un)wrapping algorithm using
    authenticated encryption *)
fun Enc(bitstring, DeviceKey): bitstring.
reduc forall m:bitstring, k:DeviceKey; Ver(
    Enc(m,k),k) = true.
reduc forall m:bitstring, k:DeviceKey; Dec(
    Enc(m, k), k) = m.
```

```
19
20 (* GroupMasterInfo and GroupMembershipInfo *)
21 table GroupMasterInfo(GroupIdentifier, bitstring,SequenceNumber, SecurityAssociationIdentifier,bitstring).
22 table GroupMembershipInfo1(GroupIdentifier, SequenceNumber, SecurityAssociationIdentifier,bitstring).
23 table GroupMembershipInfo2(GroupIdentifier, SequenceNumber, SecurityAssociationIdentifier,bitstring).
24
25 (*event*)
26 event event_GM(bitstring).
27 event event_M1(bitstring).
28 event event_M2(bitstring).
```

Line 1 describes a public communication channel among GM and devices, line 2 describes a private communication channel in GM. In the protocol, GM is divided into a mechanism called MISUser and a mechanism called MISF, but the communication between them is internal, so the communication channel is expressed using a private communication channel. Line 4 declares the secret master group key, and line 6 declares the type of the device key. Lines 7 to 10 declare the types of elements of message. Lines 12 and 13 describe a declaration for identifying the node number of the device key management tree. Lines 15 to 18 define the encryption, the verification and the decryption functions of the authenticated encryption scheme. These functions correspond to the key (un)wrapping algorithm (Wrap, Unwrap). Lines 20 to 23 define GM's list *GroupMasterInfo* and device's list *GroupMembershipInfo* for M1 and M2 by using the table functionality of ProVerif. Line 26 is GM's send event, and Lines 27 and 28 are receivers' accept event of group keys.

4.2 Formalization of Group Manager

```
1  let GM_MISUser(k0:DeviceKey, k00:DeviceKey, k01:DeviceKey, k000:DeviceKey, k001:DeviceKey, k010:DeviceKey, k011:DeviceKey) =
2
3    let CS1 = n00 in
4    let CS2 = n010 in
5    let GKD1 = Enc(mgk, k00) in
6    let GKD2 = Enc(mgk, k010) in
7    event event_GM(mgk);
8    out(cgm, (CS1, CS2, GKD1, GKD2)).
9
10 let GM_MISF() =
11   in(cgm, (CS1_:node, CS2_:node, GKD1_:bitstring, GKD2_:bitstring));
12   new GI:GroupIdentifier;
13   new SN:SequenceNumber;
14   new SAID:SecurityAssociationIdentifier;
15   out(c, (GI, SN, CS1_, CS2_, GKD1_, GKD2_, SAID));
16   insert GroupMasterInfo(GI,(CS1_,CS2_),SN, SAID,mgk).
```

Since S and T are fixed in the formalization, these inputs are omitted. Also, though *GroupMasterInfo* is inputted in the protocol in Sect. 3.2, it is omitted because *GroupMasterInfo* is managed by the table functionality of ProVerif. GM sets CS for S (lines 3 and 4), calculates the encrypted group key (lines 5 and 6), and sends a message (line 15). Lines 8 to 11 are the exchange of data between two mechanisms of GM. Line 16 is to store session information to GM's list *GroupMasterInfo*. On the first attempt, we try to use a table of ProVerif for formalizing CS. However, since the table operation of ProVerif is made private against attackers, and attacks that can be realized in practice (for example, attack which includes procedures for falsifying the contents of CS) cannot be detected, each element of CS is described as a variable.

4.3 Formalization of Devices Participating in the Group

```
let M1(k0:DeviceKey, k00:DeviceKey, k001:DeviceKey) =
  in(c, (GI':GroupIdentifier, SN':SequenceNumber, CS1':node, CS2':node, GKD1':bitstring, GKD2':bitstring, SAID':SecurityAssociationIdentifier));
  if CS1' = n0
    then (
      if Ver(GKD1', k0) = true then
      let mgk' = Dec(GKD1', k0) in
      event event_M1(mgk');
      insert GroupMembershipInfo1(GI',SN',SAID',mgk')
    )
    else if CS1' = n00
      then (
        if Ver(GKD1', k00) = true then
        let mgk' = Dec(GKD1', k00) in
        event event_M1(mgk');
        insert GroupMembershipInfo1(GI',SN',SAID',mgk')
      )
      else if CS1' = n001
        then (
          if Ver(GKD1', k001) = true then
          let mgk' = Dec(GKD1', k001) in
          event event_M1(mgk');
          insert GroupMembershipInfo1(GI',SN',SAID',mgk')
        )
        else if CS2' = n0
          then (
            if Ver(GKD2', k0) = true then
            let mgk' = Dec(GKD2', k0) in
            event event_M1(mgk');
            insert GroupMembershipInfo1(GI',SN',SAID',mgk')
          )
          else if CS2' = n00
            then (
              if Ver(GKD2', k00) = true then
              let mgk' = Dec(GKD2', k00) in
              event event_M1(mgk');
              insert GroupMembershipInfo1(GI',SN',SAID',mgk')
            )
            else if CS2' = n001
              then (
```

```
40                    if Ver(GKD2', k001) = true
                          then
41                    let mgk' = Dec(GKD2', k001)
                          in
42                    event event_M1(mgk');
43                    insert GroupMembershipInfo1(
                          GI',SN',SAID',mgk')
44                ).
```

GroupMembershipInfo is omitted because it is managed by the table functionality of ProVerif. The devices joining the group (M1 and M2) firstly receive a message from GM (line 2). After that, devices check whether CS contains the node corresponding its own by brute force (lines 3, 10, 17, 24, 31, 38), and if it is contained, decrypt the group key with the device key corresponding the node, and accept it (lines 7, 14, 21, 28, 35, 42). If it is not contained, the message is discarded. When the group key is obtained, M1 stores session information to M1's list *GroupMembershipInfo*1. This code is a formalization of the operation of M1, but other devices' formalizations are similar to M1 by adjusting the assigned device keys and CS.

4.4 Formalization of Main Process

```
1  process
2    new k0:DeviceKey;
3    new k00:DeviceKey; new k01:DeviceKey;
4    new k000:DeviceKey; new k001:DeviceKey;
         new k010:DeviceKey; new k011:DeviceKey
         ;
5
6          (*devicekeys of M3*)
7        out(c, k0);out(c, k01);out(c, k011);
8
9    ((!GM_MISUser(k0, k00, k01, k000, k001,
         k010, k011)) | (!GM_MISF()) | (!M1(k0,
          k00, k001)) | (!M2(k0, k01, k010)) |
         (!M3(k0, k01, k011)))
```

In the main process, device keys are generated and assigned according to the management tree.

4.5 Security Definition

```
1  query attacker(mgk).
2  noninterf mgk.
3  query x:bitstring; event(event_M1(x)) ==>
       event(event_GM(x)).
4  query x:bitstring; event(event_M2(x)) ==>
       event(event_GM(x)).
5  query x:bitstring; inj-event(event_M1(x))
       ==> inj-event(event_GM(x)).
6  query x:bitstring; inj-event(event_M2(x))
       ==> inj-event(event_GM(x)).
```

We verify secrecy (line 1) and strong secrecy (line 2) of the group key, and authenticity (whether the group key decrypted by the device is actually generated by GM or not). For secrecy, it is verified if there exists an event that the adversary can obtain the group key **mgk**. For strong secrecy, it is verified if the adversary can distinguish a process using the correct group key **mgk** from a process using a random group key. For authenticity, it is verified if GM always sends the message before the termination of M1 and M2. `x` is a bitstring and `event_M1(x)) ==¿ event(event_GM(x))` means correspondence of two events with the same `x`. In our verification, `x` is **mgk** or **mgk′**. Hence, only if sent **mgk** and received **mgk′** are the same, correspondence is satisfied. If the non-injective correspondence (lines 3 and 4) is true, we also verify the injective correspondence (lines 5 and 6). In addition, when verifying insider attacks, in order to represent insider attacks by M3, the device keys assigned to M3 are sent to the public communication channel (Sect. 4.4 line 7), and the attacker can use the device keys of M3.

5. Formalization of Protocol with Digital Signature Option

In this section, we formalize the mode with the digital signature option, and examine the same insider and outsider attacks. In this mode, GM sends the sequence number and the digital signature of the message. Devices verify them.

5.1 Formalization of Digital Signature Option

The definition of digital signatures is the same as the known formalization.

```
type sskey.
type spkey.
fun spk(sskey): spkey.
fun sign(bitstring, sskey): bitstring.
reduc forall m:bitstring, k:sskey; checksign
    (sign(m, k), spk(k), m) = true.
type SEQUENCENUMBER.
```

5.2 Formalization of Group Manager and Devices

We firstly show the code changes and additions of GM and devices from the formalization of the basic mode. GM make and send the signature (lines 4 and 5), devices receive and examine it (lines 2 and 3).

```
1  let GM_MISF(sskey_GM:sskey) =
2      [Section 4.2 Lines 11 to 14]
3    new sq:SEQUENCENUMBER;
4    let sig = sign((GI, SN, CS1_, CS2_, GKD1_,
          GKD2_, SAID, sq), sskey_GM) in
5    out(c, (GI, SN, CS1_, CS2_, GKD1_, GKD2_,
         SAID, sq, sig));
6    insert GroupMasterInfo(GI,(CS1_,CS2_),SN,
         SAID,mgk).
```

```
1  let M1(k0:DK, k00:DK, k001:DK, spkey_GM:
       spkey) =
2    in(c, (GI':GroupIdentifier, SN':
         SequenceNumber, CS1':node, CS2':node,
         GKD1':bitstring, GKD2':bitstring, SAID
         ':SecurityAssociationIdentifier, sq':
         SEQUENCENUMBER, sig':bitstring));
3    if checksign(sig', spkey_GM, (GI', SN',
         CS1', CS2', GKD1', GKD2', SAID', sq'))
           = true
4    then (
5          [Section  4.3  Lines  4 to 43]
6      )
```

5.3 Formalization of the Sequence Number

Next, we formalize incrementation of the sequence number. In order to verify the sequence number, GM and devices have to keep the former session's sequence number over sessions (i.e., stateful protocol). However, the variable of ProVerif is restricted to local one, and cannot be carried over to the next session. Therefore, the sequence number cannot be formalized by an obvious approach.

5.3.1 Approach with StatVerif

StatVerif [28] is a verification tool that is an extension of ProVerif to handle stateful protocols. In StatVerif, global state over multiple processes can be used, and it is possible to share the sequence number to next process by formalizing it as state. However, it is known that StatVerif does not fully support authenticity by injective correspondence. If we formalize the sequence number as state, the verification result of injective correspondence is "cannot be proved" even if we verify a simple protocol obviously satisfying injective correspondence. Hence, the mode with the digital signature option is hard to be verified with StatVerif.

5.3.2 Approach with Table Functionality

We firstly try to use the table functionality of ProVerif in order to formalize a stateful treatment of the sequence number. The table of ProVerif can transfer data between subprocess (written as process macros in ProVerif). Hence, it is expected to formalize incrementation of the sequence number. We show the code using a table to keep the sequence number.

```
1  free s0:SEQUENCENUMBER.
2  fun inc(SEQUENCENUMBER): SEQUENCENUMBER.
3  table SN_GM(SEQUENCENUMBER).
4  table SN_M1(SEQUENCENUMBER).
5
6  let GM_MISF(...) =
7      [Section 4.2 Lines 11 to 14]
8      get SN_GM(=inc(inc(inc(s0)))) in 0
9    else get SN_GM(=inc(inc(s0))) in (
10     let sq = inc(inc(inc(s0))) in
11     insert SN_GM(sq);
12     let SIGNATURE_DATA = sign(...) in
13     out(...)
14   )
15   else get SN_GM(=inc(s0)) in (
16     let sq =inc(inc(s0)) in
17     insert SN_GM(sq);
18     let SIGNATURE_DATA = sign(...) in
19     out(...)
20   )
21   else get SN_GM(=s0) in (
22     let sq = inc(s0) in
23     insert SN_GM(sq);
24     let SIGNATURE_DATA = sign(...) in
25     out(...)
26   ).
27
28 let M1(...) =
29     [Section 5.2 Lines 2 and 3]
30     get SN_M1(=inc(inc(inc(s0)))) in
31       0
32     else get SN_M1(=inc(inc(s0))) in (
33       if sq' = inc(inc(inc(s0)))
34         then (
35           insert SN_M1(sq');
36                         [Section 4.3 Lines 4 to 43]
37                 )
38     )
39     else get SN_M1(=inc(s0)) in (
40       if sq' = inc(inc(s0))
41         then (
42           insert SN_M1(sq');
43                         [Section 4.3 Lines 4 to 43]
44         )
45         else if sq' = inc(inc(inc(s0)))
46           then (
47             insert SN_M1(sq');
48                         [Section 4.3 Lines 4 to 43]
49           )
50     )
51     else get SN_M1(=s0) in (
52       if sq' = inc(s0)
53         then (
54           insert SN_M1(sq');
55                         [Section 4.3 Lines 4 to 43]
56         )
57         else if sq' = inc(inc(s0))
58           then (
59             insert SN_M1(sq');
60                         [Section 4.3 Lines 4 to 43]
61           )
62         else if sq' = inc(inc(inc(s0)))
63           then (
64             insert SN_M1(sq');
65                         [Section 4.3 Lines 4 to 43]
66           )
67           ).
```

However, ProVerif outputs "cannot be proved". This means that no concrete attack procedure is found, but ProVerif cannot prove that no attack exists. The reason seems to be contained in the branching operation by the state of tables. In ProVerif's output, an event is executed at the branch of line 57 only once. We consider that ProVerif cannot judge whether the same event will be executed again after the state of table is updated.

5.3.3 Formalization by Direct Description

As in Sect. 5.3.2, increment of the sequence number cannot be verified using authenticity of injective correspondence of ProVerif. However, by describing repeat of sessions directly and examining reachability, increment of the sequence number can be examined in order to verify replay attacks. Specifically, we formalize a situation that GM sends

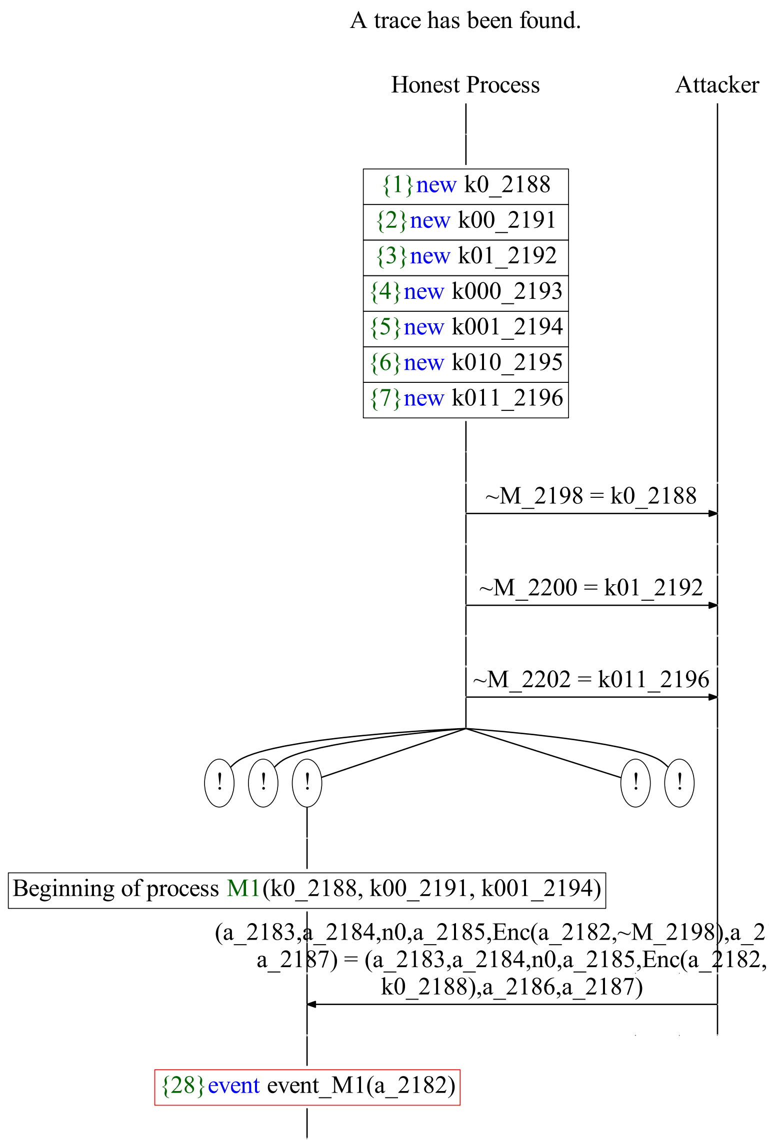


Fig. 2 Procedure of spoofing attack

the message only once (the sequence number is 1), and devices accept the group key if the received sequence number is greater than the own sequence number. So, after the first session, devices run the session again. Thus, in the second session, devices accept if the received sequence number is not 0 or 1.

We examine whether devices accept the group key twice (it means that a replay attack exists) under this situation by verifying reachability to such an event. We show the code below.

```
free s0:SEQUENCENUMBER.
fun inc(SEQUENCENUMBER): SEQUENCENUMBER.

let GM_MISF(sskey_GM:sskey) =
    [Section 4.2 Lines 11 to 14]
  let sq = inc(s0) in
  let sig = sign((GI, SN, CS1_, CS2_, GKD1_,
       GKD2_, SAID, sq), sskey_GM) in
  out(c, (GI, SN, CS1_, CS2_, GKD1_, GKD2_,
       SAID, sq, sig));
  insert GroupMasterInfo(GI,(CS1_,CS2_),SN,
       SAID,mgk).
```

```
let M1_1(...) =
    [Section 5.2]
    ->if checksign holds then M1_2(...).
let M1_2(...) =
  in(...);
  if sq' = s0 then 0
  else if sq' = inc(s0) then 0
  else if checksign(sig', spkey_GM, (GI', SN
       ', CS1', CS2', GKD1', GKD2', SAID', sq
       ')) = true
  then (
       [Section 4.3 Lines 4 to 43 (event_m1
            -> event_M1_2)]
    ).

query event(event_M1_2(mgk)).
```

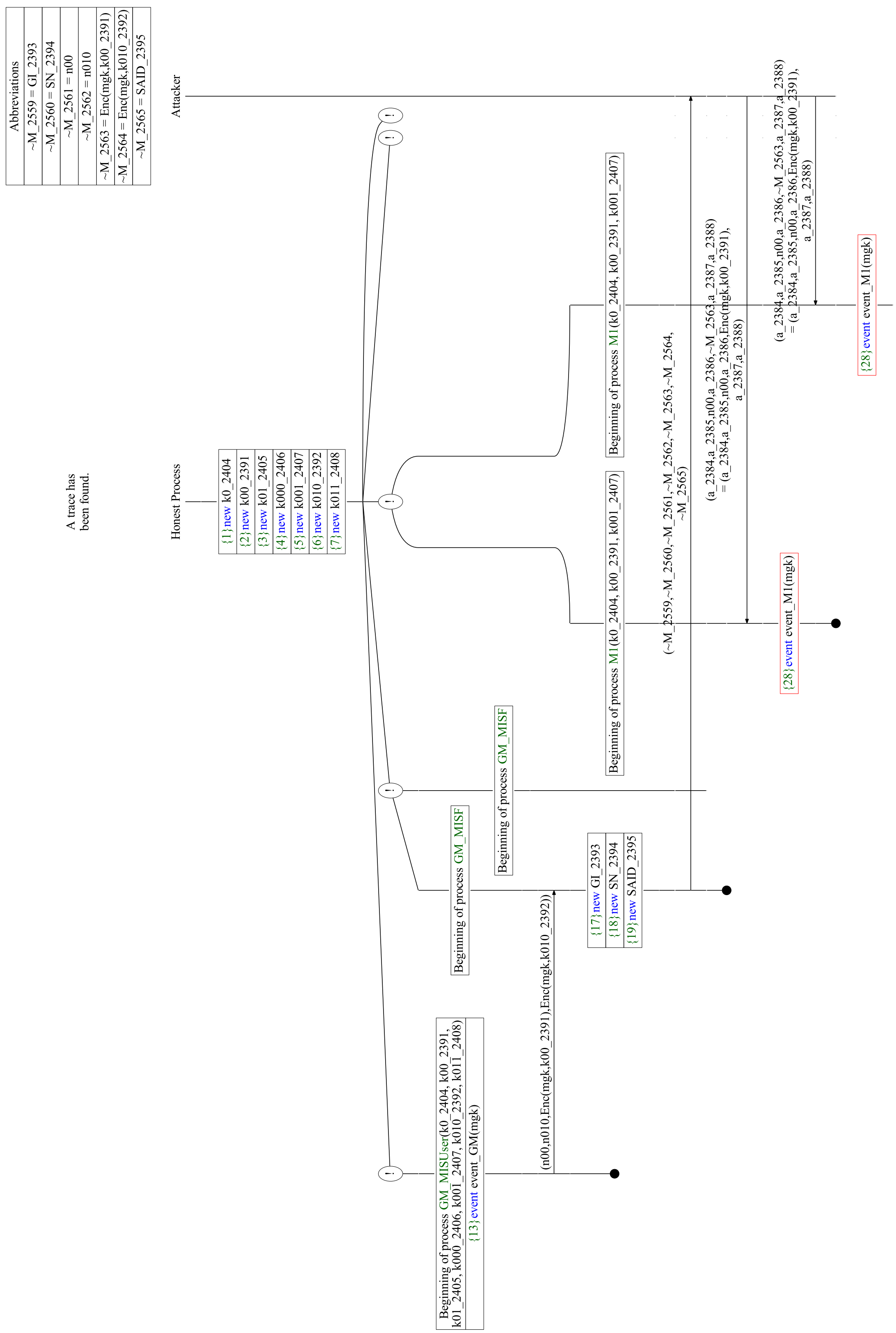


Fig. 3 Procedure of replay attack

6. Verification Result

6.1 Insider Attack

6.1.1 Without Digital Signature Option

Secrecy and strong secrecy of the group key are "true", but the authenticity is "false". The found attack is an attack that the attacker impersonates GM by using the device keys of M3. Figure 2 shows the output of ProVerif's attack procedure against M1.

We explain the found spoofing attack by using Fig. 2. First, the attacker generates a ciphertext Enc(a_2182, k0_2188) by encrypting an arbitrary value a_2182 using the device key k0_2188 shared with M1. Next, the attacker generates and sends a message containing this ciphertext. Since M1 who received it decrypts the ciphertext using k0_2188, accepts a_2182 as a group key. This corresponds to a spoofing attack by impersonating GM. In addition, a similar attack against M2 is possible with the same procedure.

6.1.2 With Digital Signature Option

All secrecy, strong secrecy and authenticity are "true". Therefore, it is clarified that the spoofing attack found for the basic mode is prevented.

6.2 Outsider Attack

6.2.1 Without Digital Signature Option

Secrecy and strong secrecy of the group key are "true", and the non-injective correspondence for authenticity is "true", but the injective correspondence is "false". The found attack is a replay attack. Figure 3 shows the output of ProVerif's attack procedure against M1.

We explain the found replay attack by using Fig. 3. First, the attacker receives a message from GM that contains the ciphertext ~M_2563 = Enc(mgk, k0_2391). When the attacker sends ~M_2563 with n00, M1 accepts the group key. Next, when the attacker sends ~M_2563 with n00 again, M1 also accepts the group key again but GM sends the message only once. This corresponds to a replay attack. In addition, a similar attack against M2 is possible with the same procedure.

6.2.2 With Digital Signature Option

Secrecy and strong secrecy of the group key are "true", and the non-injective correspondence for authenticity is "true" as well as the mode without digital signature option. As described in Sect. 5.3.3, we verify security against replay attacks by using reachability. GM sends the message only once as in Sect. 5.3.3. It formalizes the situation of a replay attack and verify if it can occur by reachability. As a result, the verification of reachability is "true". It means that event_M1_2 is not occurred, in other words, devices accept the group key only once (not twice). Thus, no replay attack exists.

7. Conclusion

For the GKM scheme in IEEE 802.21, which is specified in a HEMS specification, we verified secrecy and authenticity against insider and outsider attackers using a formal security verification tool ProVerif. It is the first verification result of IEEE 802.21 by formal methods. As a result, two attacks are found for the basic model (without the digital signature option), but these attacks are not found for the mode with the digital signature option. It means that the digital signature option correctly works to enhance security.

We formalize increment of the sequence number by using function `inc`. The current version of ProVerif supports the natural number type "nat". It is a remaining problem to formalize the sequence number by "nat".

References

[1] R. Noguchi, Y. Hanatani, and K. Yoneyama, "Verification of Group Key Management of IEEE 802.21 using ProVerif," Proc. 7th ACM Workshop on ASIA Public-Key Cryptography, pp.19–27, 2020.
[2] "IEEE 802.21-2017 - IEEE Standard for Local and metropolitan area networks–Part 21: Media Independent Services Framework," https://standards.ieee.org/standard/802_21-2017.html
[3] J. Shikata, T. Uchikoshi, M. Ebina, S. Sato, Y. Masuda, Y. Unagami, and T. Takazoe, "Security Proof of a Device Authentication Protocol for HEMS," SCIS 2018 (in Japanese).
[4] B. Blanchet, B. Smyth, V. Cheval, and M. Sylvestre, "ProVerif 2.00." http://prosecco.gforge.inria.fr/personal/bblanche/proverif
[5] "Open Specs — ECHONET," https://echonet.jp/spec_v113_lite_en/
[6] "IEC 62394:2017 - Service diagnostic interface for consumer electronics products and networks - Implementation for ECHONET," https://webstore.iec.ch/publication/32738
[7] "ISO/IEC 14543-4-3:2015 - Information technology – Home Electronic Systems (HES) architecture – Part 4-3: Application layer interface to lower communications layers for network enhanced control devices of HES Class 1," https://www.iso.org/standard/63216.html
[8] C. Cremers and M. Horvat, "Improving the ISO/IEC 11770 standard for key management techniques," Int. J. Inf. Secur., vol.15, no.6, pp.659–673, 2016.
[9] "ISO/IEC 11770-2:2018 - IT Security techniques – Key management – Part 2: Mechanisms using symmetric techniques," https://www.iso.org/standard/73207.html
[10] "ISO/IEC 11770-3:2015 - Information technology – Security techniques – Key management – Part 3: Mechanisms using asymmetric techniques," https://www.iso.org/standard/60237.html
[11] C. Cremers, "The Scyther Tool." https://people.cispa.io/cas.cremers/scyther/
[12] B. Schmidt, S. Meier, C. Cremers, and D. Basin, "Tamarin Prover." http://tamarin-prover.github.io/
[13] C. Cremers, "Key exchange in IPsec revisited: formal analysis of IKEv1 and IKEv2," Computer Security – ESORICS 2011, Lecture Notes in Computer Science, vol.6879, pp.315–334, 2011.
[14] D. Basin, C. Cremers, S. Meier, P. Degano, and J.D. Guttman, "Provably Repairing the ISO/IEC 9798 Standard for Entity Authentication," Journal of Computer Security, vol.21, no.6, pp.817–846, 2013.
[15] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, "A Comprehensive Symbolic Analysis of TLS 1.3," Proc. 2017

ACM SIGSAC Conference on Computer and Communications Security, pp.1773–1788, 2017.

[16] C. Cremers and M. Dehnel-Wild, "Component-Based Formal Analysis of 5G-AKA:Channel Assumptions and Session Confusion," Proceedings 2019 Network and Distributed System Security Symposium, 2019.

[17] C. Cremers, M. Dehnel-Wild, and K. Milner, "Secure Authentication in the Grid:A formal analysis of DNP3: SAv5," Journal of Computer Security, vol.27, no.2, pp.203–232, 2019.

[18] B. Blanchet,, "CryptoVerif: Cryptographic protocol verifier in the computational model." https://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/

[19] R. Bresciani and A. Butterfield, "ProVerif Analysis of the ZRTP Protocol," International Journal for Infonomics, vol.3, no.3, pp.306–313, 2010.

[20] S. Asadi and H.S. Shahhoseini, "Formal Security Analysis of Authentication in SNMPv3 Protocol by An Automated Tool," 6th International Symposium on Telecommunications (IST), pp.1060–1064, 2012.

[21] K. Ammayappan, "Seamless interoperation of LTE-UMTS-GSM requires flawless UMTS and GSM," 2013 2nd International Conference on Advanced Computing, Networking and Security, pp.169–174, 2013.

[22] N.B. Henda and K. Norrman, "Formal Analysis of Security Procedures in LTE - A Feasibility Study," Research in Attacks, Intrusions and Defenses, Lecture Notes in Computer Science, vol.8688, pp.341–361, 2014.

[23] J. Lu, J. Zhang, J. Li, Z. Wan, and B. Meng, "Automatic Verification of Security of OpenID Connect Protocol with ProVerif," Parallel, Grid, Cloud and Internet Computing, Lecture Notes on Data Engineering and Communications Technologies, vol.1, pp.209–220, 2017.

[24] D. Dolev and A.C.-C. Yao, "On the Security of Public Key Protocols," 22nd Annual Symposium on Foundations of Computer Science (sfcs 1981), pp.350–357, 1981.

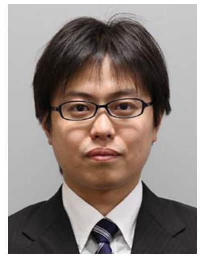
[25] D. Naor, M. Naor, and J. Lotspiech, "Revocation and Tracing Schemes for Stateless Receivers," Advances in Cryptology — CRYPTO 2001, Lecture Notes in Computer Science, vol.2139, pp.41–62, 2001.

[26] J. Dreier, M. Puys, M.-L. Potet, P. Lafourcade, and J.-L. Roch, "Formally and practically verifying flow properties in industrial systems," Comput. Secur., vol.86, pp.453–470, 2019.

[27] Y. Hanatani, N. Ogura, Y. Ohba, L. Chen, and S. Das, "Secure Multicast Group Management and Key Distribution in IEEE 802.21," Security Standardisation Research, Lecture Notes in Computer Science, vol.10074, pp.227–243, 2016.

[28] M. Ryan, M. Arapinis, and E. Ritter, "StatVerif: Verification of stateful processes." https://markryan.eu/research/statverif/

Ryoga Noguchi received the B.E. degree from Ibaraki University, Ibaraki, Japan, in 2019. He is a master's student at Ibaraki University since 2019.



Yoshikazu Hanatani received the B.E., M.E. and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 2004, 2006 and 2012, respectively. He is a researcher of Toshiba Corporation Corporate R&D Center since 2006. He is presently engaged in research on data security technologies for personal data and industrial data. He is a member of the International Association for Cryptologic Research (IACR), IEICE and JSIAM.

Kazuki Yoneyama received the B.E., M.E. and Ph.D. degrees from the University of Electro-Communications, Tokyo, Japan, in 2004, 2006 and 2008, respectively. He was a researcher of NTT Secure Platform Laboratories from 2009 to 2015. He is presently engaged in research on cryptography at the Ibaraki University, since 2015. He is a member of the International Association for Cryptologic Research (IACR), IPSJ and JSIAM.