Compact and Flexible Resolution of CBT Multicast Key-Distribution

Kanta Matsuura[1], Yuliang Zheng[2], and Hideki Imai[1]

[1] IIS, Univ. of Tokyo, Roppongi 7-22-1, Minato-ku, Tokyo 106, JAPAN
kanta@iis.u-tokyo.ac.jp, imai@iis.u-tokyo.ac.jp
[2] School of Comp. & Info. Tech., Monash Univ., Melbourne, VIC 3199, AUSTRALIA
yuliang@mars.fcit.monash.edu.au

Abstract. In an open network such as the Internet, multicast security services typically start with group session-key distribution. Considering scalability for group communication among widely-distributed members, we can find a currently-leading approach based on a CBT (Core-Based Tree) routing protocol, where Group Key Distribution Centers (GKDCs) are dynamically constructed during group-member joining process.
In search of practical use of it, this paper first analyzes the CBT protocol in terms of its efficiency as well as security management. Then the paper proposes several improvements on the protocol with an aim to solve the problem identified. In particular, (1) an overuse of encryption and signatures is avoided and (2) a hybrid trust model is introduced by a simple mechanism for controling the GKDC distribution. A comprehensive comparison among the costs of several implementations is also carried out.

1 Introduction

Multicast-oriented applications require a sufficient security infrastructure especially when implemented in an open and global network. A good example is the Internet, where the next-generation protocol IPv6 (Internet Protocol version 6) considers security services for multicast as one of the central issues [**?**], [**?**]. The basic starting-point is secure and authenticated distribution or agreement of group session-keys.

A simple strategy is to assign the key-distribution function to a trusted single entity or Key Distribution Center (KDC). This strategy, however, very unlikely scales for multicast communication among widely- or sparsely-distributed members. Scalable approaches would be combined with multicast routing protocols since

- there exist routing protocols which provide dynamic and scalable properties,
- routing mechanisms are typically in close relation to group structures,

and

- combining two pre-processes (routing preparation and key-distribution) potentially saves bandwidth.

Looking at the Internet, we can find IGMP (Internet Group Management Protocol) [?] used in the final delivery of a multicast packet between a local router and a group member on its directly-attached subnetworks. IGMP delivery services can be jointed by a number of different multicast routing and delivery mechanisms among distributed routers.

For groups with dispersed or sparse membership, most scalable are Shared-Tree techniques such as PIM-SM (Protocol-Independent Multicast – Sparse Mode) [?] and CBT (Core-Based Tree) routing protocol [?]–[?]. The main differences between these two techniques are that

- CBT maintains its characteristics as scalable as possible by not offering the option of shifting from a Shared Tree to a Shortest Path Tree

and that

- CBT has fewer entries in the routing tables [?].

Thus, although not devoted to any specific implementation, a dynamic key-distribution protocol in conjunction with CBT [?] is currently considered as a strong candidate for a scalable multicast key-distribution scheme. This protocol uses Group Key Distribution Centers (GKDCs) which are dynamically constructed during group-member joining process.

This paper first overviews the CBT mechanism in Sect. 2, where several problems or questions are subsequently discussed, and finally four implementations are evaluated in terms of their computational costs and communication overhead. A more compact and yet more flexible resolution is then proposed in Sect. 3 with a function of controlling GKDCs, followed by the evaluation of four implementations. After discussion in Sect. 4, Sect. 5 gives conclusions.

2 Core-Based Tree

2.1 Routing Protocol

In the CBT routing protocol [?]–[?], a single shared delivery tree is built around several core routers. When a host wishes to join the multicast group, it casts an IGMP group membership report across its attached link. On receiving this report, a local CBT-aware router explicitly joins the delivery tree by generating a `JOIN_REQUEST` message, which is sent to the next hop on the path towards one of the group's core routers. In reply to this `JOIN_REQUEST` message, a `JOIN_ACK` message is generated by the core or another router which has already joined the tree on the path between the core and the host. This `JOIN_ACK` message traverses the reverse path of the join and thus a new branch is created. Routers along the new branch are called non-core routers, and there exists a parent-child relationship between adjacent routers along the branch. The resulting tree is a bidirectional and acyclic graph that reaches every member host. Once the tree is established, packets are forwarded in a simple way: when a node receives a

multicast packet, it forwards copies of the packet on all branches of the group's tree except for the branch on which the packet arrived.

There are two significant differences between the CBT and other network-layer multicast routing protocols such as DVMRP (Distance Vector Multicast Routing Protocol) and M-OSPF (Multicast extensions for Open Shortest Path First) [**?**]. First, different from DVMRP and M-OSPF, CBT messages have their own protocol headers, which allow the protocol to make explicit provision for security. Second, CBT is a Shared-Tree technique; there exists one shared delivery tree per group, not per sender. By contrast, the other protocols use Source-Based Tree architecture, which is less scalable than Shared-Tree architecture.

2.2 Security Mechanism

The CBT architecture complements security by secure joining process [**?**]. This complementation assumes the presence of an internetwork-wide asymmetric cryptosystem[3]. In particular, join-process messages such as IGMP group membership reports, `JOIN_REQUEST` messages, and `JOIN_ACK` messages are equipped with security parameters by the use of the asymmetric cryptosystem.

We describe an example of secure CBT joining by using a sample tree shown in Fig. 1[4]. The main terms used in the description are as follows.

C: An elected primary core which takes on the initial role of GKDC.
TKN-X: Token of an entity X, typically containing recipient's identity, a time-stamp denoted as TS-X, and a pseudo-random number to help a recipient with verification procedures; the time-stamp demonstrates message freshness, and the random number demonstrates message originality. If X is a host, recipient's identity is omitted.
$\boldsymbol{SIG_X(\cdot)}$**:** Signature with the secret key of X.
$\boldsymbol{ENC_X(\cdot)}$**:** Encryption with the public key of X.
KM: Key materials to be delivered. A group key and a key-encrypting key are contained. The former is used for encrypting group data traffic. The latter is created by the primary core and used for re-keying the group with a new group key just prior to an old key exceeding its lifetime.
ACL: Group access control list created by a CBT group initiator.
SAp: Security Association parameters pre-negotiated through ISAKMP (Internet Security Association and Key Management Protocol) [**?**] or Photuris Key Management Protocol [**?**].
GAP: Group access package sent from an already-verified node to a joining node.

The format of GAP is

$$SIG_{\text{sender}}\left(\text{TKN-sender}, SIG_{\text{C}}\left(\text{ACL}\right), SE_{\text{C}\rightarrow\text{host}}, SE_{\text{C}\rightarrow\text{next-hop}}\right),$$

[3] Public-key management systems can be globally provided, for instance, through extended DNS (Domain Name System) [**?**] or WWW (World Wide Web) [**?**].
[4] This tree has only one core for simplicity, but typically a CBT tree comprises several.

where $SE_{\mathrm{X}\to\mathrm{Y}} = ENC_{\mathrm{Y}}\left(SIG_{\mathrm{X}}\left(\mathrm{KM},\mathrm{SAp}\right)\right)$.
ACL is assumed to have been already sent from the initiator to the primary core. It is also assumed that the primary core has already participated in Security Association establishment to hold KM and SAp. These pre-processes are carried out by using a unicast protocol such as ISAKMP [**?**] or Photuris [**?**].

Join-Request: First, the host h sends an IGMP group membership report

$$\left(SIG_{\mathrm{h}}\left(\text{TKN-h}\right), \texttt{MEMBERSHIP\_REPORT}\right)$$

to the local router A. On receiving this report, A authenticates the h's token which is digitally-signed. If successful, A generates a CBT join-request

$$SIG_{\mathrm{A}}\left(\text{TKN-A}, SIG_{\mathrm{h}}\left(\text{TKN-h}\right), \texttt{JOIN\_REQUEST}\right)$$

and unicasts it to the next-hop router B on the path to the core C. Next, B verifies A's signature. If successful, B relays the request to the core C by unicasting

$$SIG_{\mathrm{B}}\left(\text{TKN-B}, SIG_{\mathrm{h}}\left(\text{TKN-h}\right), \texttt{JOIN\_REQUEST}\right).$$

Finally, the core C verifies B's signature and also h's signature. If this is successful and the host h is found to be included in the group access control list ACL, C generates a group access package GAP.

Join-Acknowledgement: The core router encapsulates the group access package GAP in a join-acknowledgement, and digitally signs the whole message:

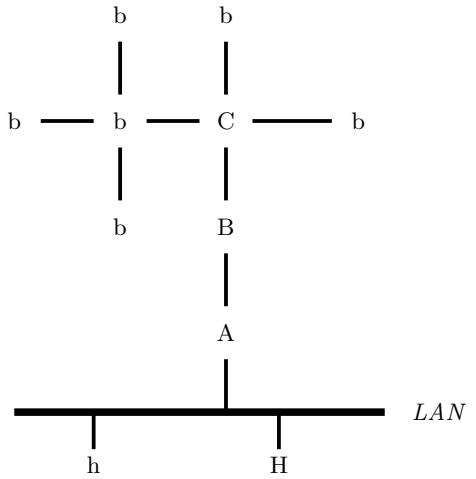


Fig. 1. An example of Core-Based Tree. C is the primary core. A, B, and b's are non-core routers. h is a host which wishes to join the multicast group. The LAN (Local Area Network) where h resides is under its local CBT-aware router A. Another host H is on the same LAN.

$$SIG_{\mathrm{C}}\left(SIG_{\mathrm{h}}\left(\text{TKN-h}\right), \text{GAP}, \texttt{JOIN\_ACK}\right).$$

This is sent to B. Then C's signatures on the whole message and on GAP are verified by B. If both successful, B extracts the encrypted information $SE_{\mathrm{C}\to\mathrm{B}}$ from GAP and decrypts it. Then B verifies C's signatures on this information and on the access control list ACL. If successful, B subsequently stores the ACL in an appropriate table, encrypts the information (key materials and security-association parameters) with a local key, and also stores it. Next, B encrypts the key information with the next-hop router A's public key to reform GAP. The resulting whole acknowledgement message

$$SIG_{\mathrm{B}}\left(SIG_{\mathrm{h}}\left(\text{TKN-h}\right), \text{GAP}, \texttt{JOIN\_ACK}\right)$$

is sent to A. On receiving this message, A follows the same verification and storage process. Subsequently, A generates an IGMP group membership report

$$\left(SIG_{\mathrm{h}}\left(\text{TKN-h}\right), SE_{\mathrm{C}\to\mathrm{h}}, \texttt{MEMBERSHIP\_REPORT}\right),$$

where $SE_{\mathrm{C}\to\mathrm{h}}$ is extracted from the received GAP and just forwarded. The host h receives this report and identifies it by its signed token. Finally, h decrypts and verifies the key information. Thus h securely obtains the key materials and the security-association parameters, and the routers A and B become GKDCs[5].

Once the above process is successfully completed, the router A, which is now available as GKDC, directly responds to the request

$$\left(SIG_{\mathrm{H}}\left(\text{TKN-H}\right), \texttt{MEMBERSHIP\_REPORT}\right)$$

from another host H by sending

$$\left(SIG_{\mathrm{H}}\left(\text{TKN-H}\right), SE_{\mathrm{C}\to\mathrm{H}}, \texttt{MEMBERSHIP\_REPORT}\right)$$

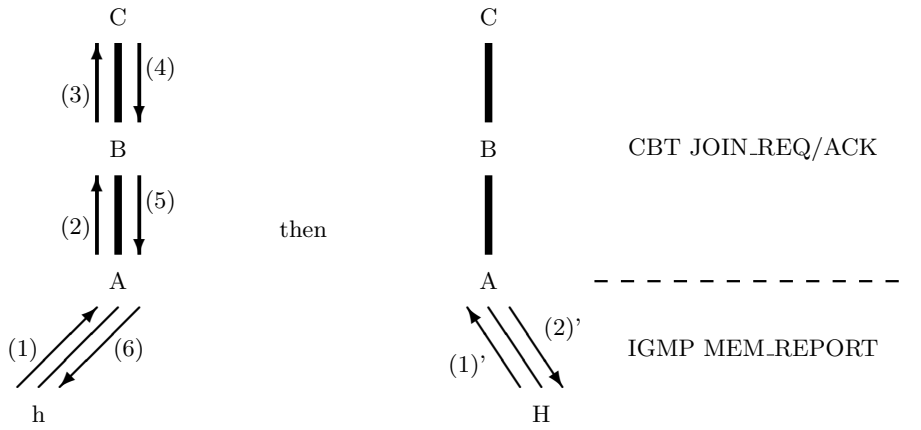
back to H. $SE_{\mathrm{C}\to\mathrm{H}}$ can be generated by A, since A knows $SIG_{\mathrm{C}}\left(\text{KM}, \text{SAp}\right)$. H then identifies it by its signed token and decrypts $SE_{\mathrm{C}\to\mathrm{H}}$. Finally, if the core's signature on the decrypted information is successfully verified, the report is accepted. The whole process is summarized in Fig. 2.

2.3 Problems and Questions

Security: First, information on key can be distributed widely, which might provide less security for the system; uncontrollable scalability could be a threat. The system trusts each router's behavior according to the spirit of the Internet, but this trust does not automatically mean that the data-storage systems of the routers are sufficiently protected against attackers.

Second, GKDCs store key materials in encrypted form, but without any freshness parameters. This is less secure than storage with freshness parameters such as time-stamps or nonces, which are deterrents to replay attacks.

[5] When any process failure occurs or an invalid signature is found, a `JOIN_NACK` message is returned toward the local router A. The host h is then notified of the failure by a resultant IGMP membership report. It initiates a join-request again if it wishes.



(1): SIG_{h} (TKN-h)
(2): SIG_{A} (TKN-A, SIG_{h} (TKN-h))
(3): SIG_{B} (TKN-B, SIG_{h} (TKN-h))
(4): $SIG_{\mathrm{C}}\big($ SIG_{h} (TKN-h), $\underline{\boldsymbol{SIG}_{\mathrm{C}}(\text{TKN-C}, SIG_{\mathrm{C}}(\text{ACL}),}$
$\underline{ENC_{\mathrm{h}}(SIG_{\mathrm{C}}(\text{KM, SAp}))}, ENC_{\mathrm{B}}(SIG_{\mathrm{C}}(\text{KM, SAp})))\big)$
(5): $SIG_{\mathrm{B}}\big($ SIG_{h} (TKN-h), $\underline{\boldsymbol{SIG}_{\mathrm{B}}}(\text{TKN-B}, SIG_{\mathrm{C}}(\text{ACL}),$
$\underline{ENC_{\mathrm{h}}(SIG_{\mathrm{C}}(\text{KM, SAp}))}, ENC_{\mathrm{A}}(SIG_{\mathrm{C}}(\text{KM, SAp})))\big)$
(6): SIG_{h} (TKN-h) , ENC_{h} ($\underline{SIG_{\mathrm{C}}}$ (KM, SAp))
(1)': SIG_{H} (TKN-H)
(2)': SIG_{H} (TKN-H) , ENC_{H} (SIG_{C} (KM, SAp))

Fig. 2. Process description of secure joining to the sample Core-Based Tree. After the joining process of host h, another host H makes a join-request.

Efficiency: First, sender's signature on GAP (highlighted by underlined fonts in Fig. 2) is redundant and unnecessary. This is because GAP is carried by a join-acknowledgement message which is signed as a whole by the sender.

Second, the first signed and encrypted information, which is underlined in Fig. 2, might be also redundant; the same information is contained in the following part. If really redundant, this underlined part is considered to cause a waste of bandwidth; in general, simultaneous use of digital signature and public-key encryption leads to large communication overhead.

Digital signature and public-key encryption cost a lot not only in bandwidth but also in computation. The cost depends on cryptographic algorithms, which motivates us to evaluate the efficiency of specific implementations[6].

[6] No specific implementation is discussed in the Internet RFC (Request for Comments) [**?**] which proposes the CBT-based key-distribution protocol.

2.4 Evaluation of Implementation

This subsection evaluates the efficiency of the following four implementations[7]:

`RSA:` $SIG(\cdot)$ = RSA signature, $ENC(\cdot)$ = RSA encryption.
`ElGamal:` $SIG(\cdot)$ = SDSS, $ENC(\cdot)$ = ElGamal encryption.
`SC:` $SIG(\cdot)$ = SDSS, $ENC_{\text{router}}(\cdot)$ = signcryption,
$SE_{\text{C}\to\text{host}}(\cdot)$ = signcryption.
`SC_RSA:` $SIG(\cdot)$ = RSA signature, $ENC_{\text{router}}(\cdot)$ = signcryption,
$SE_{\text{C}\to\text{host}}(\cdot)$ = signcryption.

Since signcryption cannot change the recipient with keeping the signer, the last part of GAP in `SC`/`SC_RSA` uses signcryption just as an encryption scheme.

Computational Cost: We estimate the cost of public-key cryptographic computation required by one execution of join-request and join-acknowledgement process in each implementation. For specific comparison, we have to consider the currently-required size of the exponents in those schemes, since the computational cost is mainly determined by the size of the exponent.

With RSA, the main computational cost is in decryption or signature generation which generally involves a modular exponentiation with a *full size* exponent, namely about $1.5|n|$ modular multiplications using the "square-and-multiply" method, where n indicates the RSA composite involved and $|n|$ denotes the size or length (in bits) of n. With the help of the Chinese Remainder Theorem, this cost can be reduced to be $1.5|n|/4 = 0.375|n|$. On the assumption that $|n| = 1536$, which is recommended to be used for long-term security (say, more than 20 years), we compute the total cost of the implementation `RSA`.

Next, `ElGamal` requires (1) one modular exponentiation for generating a signature, (2) two for verifying a signature, (3) two for encrypting, and (4) one for decrypting. Resulting numbers of modular multiplications are (1) $1.5|q|$, (2) $1.75|q|$, (3) $3|q|$, and (4) $1.5|q|$, respectively, where q is the order of the subgroup used in SDSS. Assuming that $|q| = 176$, which provides long-term security quite similar to that considered in the case of `RSA`, we compute the total computational cost of `ElGamal`.

Likewise, we compute the costs of `SC` and `SC_RSA`; (1) one modular exponentiation for signcrypting and (2) two for unsigncrypting result in (1) $1.5|q|$ and (2) $1.75|q|$ modular multiplications. Signature generation and verification cost the same as in `ElGamal` and in `RSA`, respectively.

Those costs are given as a function of the number of non-GKDC routers, as shown in Fig. 3. `SC_RSA` is the most efficient and `ElGamal` is the least efficient. The cost of `SC_RSA` is about 74% of that of `ElGamal` when there exist non-GKDC routers.

[7] Since signcryption is a relatively new cryptographic primitive, we overview it in Appendix. The readers can consult textbooks (for example, [**?**]) about the other well-known encryption/signature algorithms.

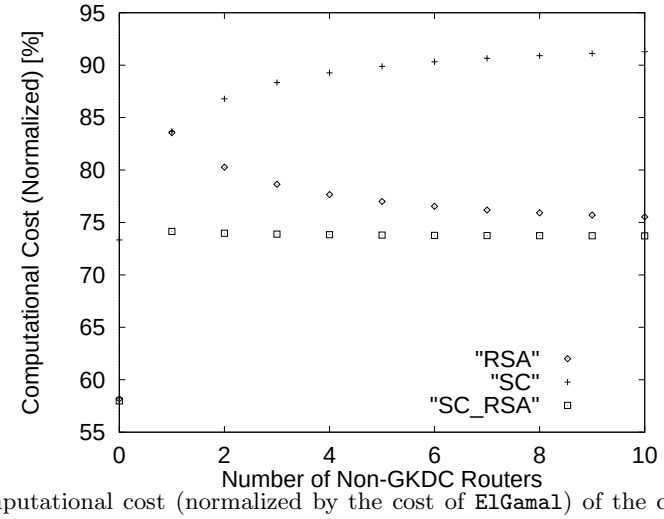


Fig. 3. Computational cost (normalized by the cost of `ElGamal`) of the conventional CBT scheme with $|n| = 1536$ and $|q| = 176$.

Table 1. Communication overhead of `SC` represented by the ratios to those of `RSA`, `ElGamal`, and `SC_RSA`.

	(h → A) or (H → A)	(A → B) or (B → C)
vs. `RSA`	17.2%	17.2%
vs. `ElGamal`	100.0%	100.0%
vs. `SC_RSA`	17.2%	17.2%

(a) during join-request process

	(C → B) or (B → A)	(A → h) or (A → H)
vs. `RSA`	15.0%	11.5%
vs. `ElGamal`	39.7%	25.6%
vs. `SC_RSA`	22.5%	29.3%

(b) during join-acknowledgement process

Communication Overhead: Assuming the same security level as in the evaluation of computational cost, we use $|p| = |n| = 1536$ and $|hash(\cdot)| = |q|/2 = 88$, where p is the order of the multiplicative group and $hash(\cdot)$ is the one-way hash function used in SDSS-based schemes. The evaluation result is shown in Table 1. Since `SC` is estimated to be the most compact, this result is represented by the ratios of the overhead of `SC` to those of the other implementations.

3 An Efficient Resolution

To solve or answer to the issues identified in Sect. 2.3, this section describes an efficient resolution of secure CBT key-distribution.

3.1 Protocol

The resolution is simple enough to keep the advantages of the original scheme; just the format of GAP is changed into

$$\left(\text{TKN-sender}, SIG_{\text{C}}\left(\text{ACL}\right), \text{TTL}, ENC_{\text{nd}}\left(SIG_{\text{C}}\left(\text{KM}, \text{SAp}\ \left[, \text{TS-C}\ \right]\ \right)\right)\right)$$

where "nd" is the node which can decrypt the encrypted information next. [ ] denotes an option and TTL is a Time-to-Live parameter for GKDC permission; as described later, nd=next-hop if TTL is positive, and otherwise nd=host. The core assigns an initial value to TTL according to its security policy, which will be discussed later in Sect. 4. Different from the conventional CBT protocol, GAP itself is not signed.

In particular, after exactly the same join-request process as in the original CBT scheme, the core assigns a non-negative value to TTL, and generates a join-acknowledgement message according to the GAP format refined above. This message is forwarded toward the host hop by hop. On receiving the join-acknowledgement message, each non-local router first verifies the sender's signature on the whole message[8]. If successful, one of the following three procedures occurs, depending on the value of TTL[9]:

(TTL>1) The router decrements the TTL and subsequently decrypts or unsigncrypts the information contained in the last part of GAP. Then the core's signatures on this information and on ACL are verified. If successful, the router encrypts (or signcrypts) the information with the public key of the next-hop (and additionally with the router's own secret key in the case of signcryption). The resultant acknowledgement message as a whole is signed with the router's secret key and forwarded. The router stores KM, SAp, and ACL in a signed and encrypted form to become GKDC. This stored information optionally includes the time-stamp generated by the core, depending on the security policy.

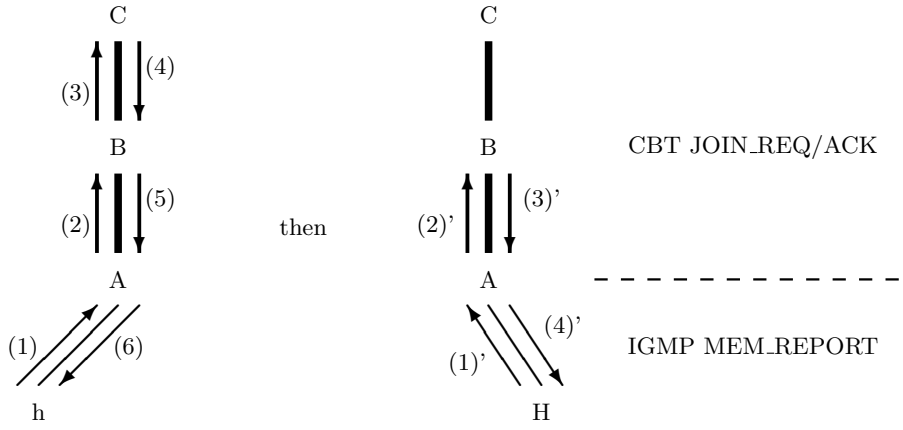
(TTL=1) The router decrements the TTL and subsequently decrypts or unsigncrypts the last part of GAp. Then the core's signatures are verified. In addition, the host's signature on the token is verified with the public key consistently derived from the CBT routing information. If everything is successful, the router encrypts (or signcrypts) the information with the public key of the host (and additionally with the router's own secret key in the case of signcryption). The resultant acknowledgement message is forwarded in a digitally-signed form. The storage process occurs in the same way as in the case of TTL>1.

(TTL=0) The router is not qualified as GKDC; neither key materials nor ACL is stored. The router just signs the whole message and forwards it to the next-hop.

---

[8] When any joining-process failure occurs or an invalid signature is found, a `JOIN_NACK` process is provoked in the same way as in the conventional scheme.

[9] If an attacker changes the value of TTL, it can be detected by the verification of the sender's signature on the whole message.



(1): SIG_{h} (TKN-h)
(2): SIG_{A} (TKN-A, SIG_{h} (TKN-h))
(3): SIG_{B} (TKN-B, SIG_{h} (TKN-h))
(4): SIG_{C}(SIG_{h} (TKN-h), TKN-C, SIG_{C} (ACL) , TTL, ENC_{B} (SIG_{C} (KM, SAp [, TS-C])))
(5): SIG_{B}(SIG_{h} (TKN-h), TKN-B, SIG_{C} (ACL) , TTL, ENC_{h} (SIG_{C} (KM, SAp [, TS-C])))
(6): SIG_{h} (TKN-h) , ENC_{h} (SIG_{C} (KM, SAp [, TS-C]))
(1)': SIG_{H} (TKN-H)
(2)': SIG_{A} (TKN-A, SIG_{H} (TKN-H))
(3)': SIG_{B}(SIG_{H} (TKN-H), TKN-B, SIG_{C} (ACL) , TTL, ENC_{A} (SIG_{C} (KM, SAp [, TS-C])))
(4)': SIG_{H} (TKN-H) , ENC_{H} (SIG_{C} (KM, SAp [, TS-C]))

Fig. 4. Process description of the proposed secure joining to Core-Based Tree when the initial value of TTL is 1. The joining process of host h qualifies router B as a GKDC, and then that of host H qualifies A.

The local router follows the same process, except that the resultant message is not a join-acknowledgement message but an IGMP group membership report

$$\left(SIG_{\mathrm{h}}\,(\text{TKN-h})\,, ENC_{\mathrm{h}}\left(SIG_{\mathrm{C}}\,(\text{KM}, \text{SAp }\ [, \text{TS-C }]\ )\right), \texttt{MEMBERSHIP\_REPORT}\right).$$

The whole process is exemplified in Fig. 4.

3.2 Evaluation of Implementation

This subsection evaluates the proposed protocol considering four implementations in a similar way as in Sect. 2:

`RSA*:` $SIG(\cdot)$ = RSA signature, $ENC(\cdot)$ = RSA encryption.
`ElGamal*:` $SIG(\cdot)$ = SDSS, $ENC(\cdot)$ = ElGamal encryption.
`SC*:` $SIG(\cdot)$ = SDSS, $ENC(\cdot)$ = signcryption.
`SC_RSA*:` $SIG(\cdot)$ = RSA signature, $ENC(\cdot)$ = signcryption.

For simplicity, the initial value of TTL is assumed to be large enough for all the routers to become GKDCs through one joining (Fig. 4 is not the case).

Computational Cost: The same security level as in Sect. 2 is considered. The evaluation result is shown in Fig. 5; `SC_RSA*` is the most efficient. Even the least efficient one, `ElGamal*`, is more efficient than the corresponding implementation (`ElGamal`) of the conventional protocol. Figure 6 illustrates how much the proposed protocol reduces the computational cost in comparison with the conventional protocol reviewed in Sect. 2.

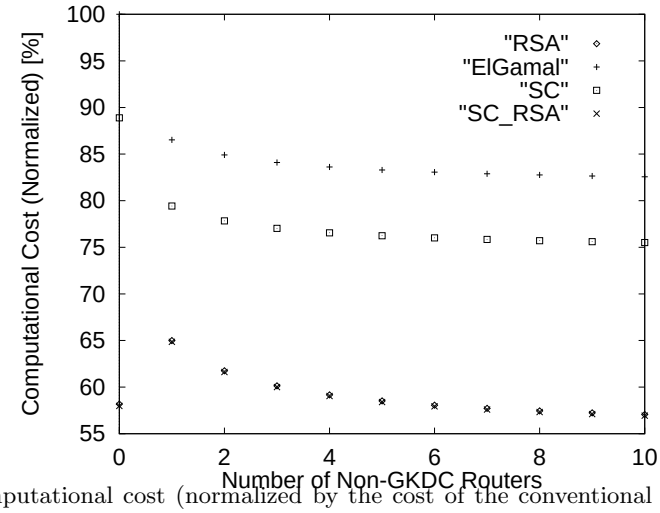


Fig. 5. Computational cost (normalized by the cost of the conventional `ElGamal`) of the proposed scheme with $|n| = 1536$ and $|q| = 176$.

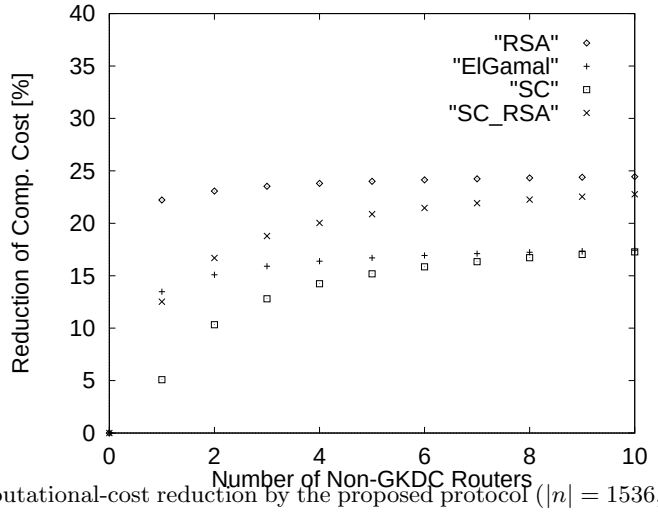


Fig. 6. Computational-cost reduction by the proposed protocol ($|n| = 1536$, $|q| = 176$).

Table 2. Communication overhead during join-acknowledgement process of `SC*` represented by the ratios to those of `RSA*`, `ElGamal*`, and `SC_RSA*`. Considered parameters are: $|p| = |n| = 1536$ and $|hash(\cdot)| = |q|/2 = 88$.

	(C → B) or (B → A)	(A → h) or (A → H)
vs. `RSA*`	17.2%	17.2%
vs. `ElGamal*`	50.9%	38.4%
vs. `SC_RSA*`	20.6%	23.7%

Communication Overhead: There is no difference between the conventional and the proposed protocols, in the overhead during join-request process. Table 2 shows the communication overhead during join-acknowledgement process, where the same security level as in previous discussion is assumed. Since `SC*` is estimated to be the most compact, the comparison result is represented by the ratios of the overhead of `SC*` to those of the other implementations.

In comparison with the original protocol, the proposed protocol requires shorter communication overhead during the join-acknowledgement process; assuming $|\mathrm{TTL}| \ll |KM|$, $|\mathrm{TS\text{-}C}| \ll |KM|$, and the same security level as in the previous discussions ($|p| = |n| = 1536$ and $|hash(\cdot)| = |q|/2 = 88$), the overhead reduction of the messages between routers is estimated as follows:

$$\texttt{RSA*} \text{ vs. } \texttt{RSA}: \quad \frac{3|n| + |\mathrm{KM}| + |\mathrm{SAp}|}{8|n| + |\mathrm{KM}| + |\mathrm{SAp}|} > 37.5\%,$$

$$\texttt{ElGamal*} \text{ vs. } \texttt{ElGamal}: \quad \frac{2\left(|hash\left(\cdot\right)| + |q|\right) + |p| + |\mathrm{KM}| + |\mathrm{SAp}|}{6\left(|hash\left(\cdot\right)| + |q|\right) + 2|p| + |\mathrm{KM}| + |\mathrm{SAp}|} > 44.3\%,$$

$$\texttt{SC*} \text{ vs. } \texttt{SC}: \quad \frac{2|hash(\cdot)| + 2|q| + |\mathrm{KM}| + |\mathrm{SAp}|}{7|hash(\cdot)| + 7|q| + |\mathrm{KM}| + |\mathrm{SAp}|} > 28.6\%,$$

and

$$\texttt{SC\_RSA*} \text{ vs. } \texttt{SC\_RSA}: \quad \frac{|n| + |q| + |hash(\cdot)| + |\mathrm{KM}| + |\mathrm{SAp}|}{5|n| + 2|q| + 2|hash(\cdot)| + |\mathrm{KM}| + |\mathrm{SAp}|} > 21.9\%.$$

4 Discussion

4.1 A Question of Trust

The CBT security architecture assumes that all the routers on a delivery tree are trusted and do not misbehave. In reply to the question whether this assumption is reasonable in internetworking, the original RFC (Request for Comments) [**?**] makes two remarks:

(A1) Depending on the security requirement and perceived threat, the presented model may be acceptable.

(A2) A higher level of security can be provided by imposing all the join-request authentication on a core router.

In our scheme, the latter (A2) can be provided by assigning the minimum initial value to Time-to-Live parameter, *i.e.*, TTL=0. The flexibility of this kind of control with the help of TTL will be discussed in the next subsection. It should be noted that even this "only one GKDC" situation is still more scalable than protocols with "only one KDC"; different groups can use different GKDCs in CBT-based protocols.

The proposed scheme provides another improvement on security by optionally including an authenticated time-stamp in the key-material container. Such a use of a time-varying quantity convinces the entities that the session-keys are *fresh*. As well-known in the research community of information-security technologies, freshness of the keys contributes to the protection against replay attack. This time-stamp can be replaced with other freshness parameters such as nonces.

4.2 Controlling GKDC Distribution

A need for controlling GKDC distribution or expansion is suggested by the higher-level security modification (A2) mentioned in the previous subsection. This control should be viewed not only from security requirements, but also from load distribution/concentration and traffic efficiency. A large initial value of TTL precludes too much burden from concentrating on a small number of GKDC routers, but might provide insufficient security. By contrast, a small initial value of TTL provides us a high level of security, but might cause too heavy load on GKDC routers and traffic flood. One practical solution to this balancing problem is to offer a flexible controlling mechanism *with a cost as small as possible*; we propose the use of a Time-to-Live parameter TTL, which permits the recipient router to become GKDC when positive. Both computational cost and communication overhead caused by the deployment of TTL are negligible.

The use of TTL implies a hybrid trust model; the trust tree is no longer the same as the distribution tree. The core assigns an initial value to TTL to show the region of the routers which the core itself trusts. Suppose that the router B receives a join-acknowledgement message with TTL=1 from the core C, which tells B that B is trusted by C but that farther downstream routers are not trusted by C. This acknowledgement process does not allow the farther routers to become GKDCs this time. When the next join-acknowledgement process occurs as a result of the next join-request, B assigns an appropriate initial value to TTL to show the region of the routers which B trusts; if the next-hop router A is trusted by B, B can assign a positive initial value without consulting the core C. Thus the trusted region can be expanded request by request.

4.3 Sender-Specific Keys

After completion of the proposed key-distribution, each member can easily multicast a sender-specific key by signing it with his/her own secret key and encrypting the result with the obtained group key. This message includes necessary Security Association parameters.

Non-member senders can also distribute a sender-specific key. They first negotiate with the primary core to establish the Security Association parameters and their session key. Thereafter, by using the group's current session-key, the core multicasts the sender-specific session-key together with the sender's security parameters to the group.

5 Conclusions

This paper analyzed and improved a scalable multicast key-distribution based on CBT.

In the analysis, four implementations were compared: RSA-based scheme (`RSA`), ElGamal-based scheme (`ElGamal`), signcryption-based scheme (`SC`), and signcryption-with-RSA scheme (`SC_RSA`). In cryptographic computation, `SC_RSA` costs least and `ElGamal` costs most. The computational cost of `SC` is moderate, and its communication overhead is by far the shortest. The computational cost was examined in detail as a function of the number of non-GKDC routers between the joining host and the core router of the group.

Likewise, in the proposed protocol, four implementations were compared: `RSA*`, `ElGamal*`, `SC*`, and `SC_RSA*`. The result was quite similar to that of the conventional protocol; `SC_RSA*` gives the smallest computational cost, and `SC*` gives the shortest communication overhead. In each implementation, the proposed protocol is more efficient than the conventional one. The computational-cost reduction depends on the number of non-GKDC routers and saturates to be approximately 24% (`RSA*` vs. `RSA`, or `SC_RSA*` vs. `SC_RSA`) and 17% (`ElGamal*` vs. `ElGamal`, or `SC*` vs. `SC`). The overhead is also reduced significantly; at least 37.5% saving in the case of `RSA*`, 44.3% in the case of `ElGamal*`, 28.6% in the case of `SC*`, and 21.9% in the case of `SC_RSA*`. These advantages of the proposed protocol are due to the deployment of a cheaper Group Access Package than the conventional one.

To summarize the issues on efficiency, the combination of RSA and signcryption is the best implementation in terms of the computational cost and the signcryption-based implementation is the best way in terms of the communication overhead, both in the original protocol and in the proposed protocol. For each implementation with more than two non-GKDC routers, the proposed protocol saves at least 13% of the computational cost and 22% of the communication overhead.

In addition to the efficiency, the proposed scheme provides two improvements. The first one is a function of controlling the expansion or distribution of GKDCs. This contributes to protocol flexibility in balancing security requirements and traffic efficiency or load concentration. The second one is a freshness parameter optionally included in a returned key-material container. This contributes to the protection against replay attack.

Appendix: SDSS and Signcryption

To avoid forgery and ensure confidentiality of the contents of a letter, for centuries it has been a common practice for the originator of the letter to sign his/her name on it and then seal it in an envelope, before posting it. In a secure and authenticated communication over an open and insecure network, the same two-step approach has been followed; before a message is sent out, the sender of the message would sign it using a digital signature scheme, and then encrypt the message (and the signature) using a private-key encryption algorithm under a randomly chosen message-encryption key. The random message-encryption key would then be encrypted by using the recipient's public key. We call this two-step approach *signature-then-encryption*.

In general, the sum of the cost for signature and the cost for public-key encryption is computationally expensive. Originating from questioning whether it is absolutely necessary for one to spend the sum of the costs to achieve both confidentiality and authenticity, the author of [**?**] proposes a primitive called *signcryption*.

Intuitively, a digital signcryption is a cryptographic method that fulfills both the functions of secure encryption and digital signature, but *with a cost smaller than that required by signature-then-encryption*. This paper uses an implementation of signcryption based on a Shortened Digital-Signature Standard (SDSS). By using the notation listed in Table 3, Fig. 7 shows an SDSS which is referred to as SDSS1 in [**?**]. The signcryption based on this SDSS1 is illustrated in Fig. 8.

Table 3. Notation.

Parameters public to all:
p — a large prime
q — a large prime factor of $p-1$
g — an integer with order q modulo p chosen randomly from $[1, 2, \cdots, p-1]$
Keys of the sender and the recipient:
$SK_{\text{Alice}} \in [1, 2, \cdots, q-1]$ — secret key of Alice
PK_{Alice} — public key of Alice where $PK_{\text{Alice}} = g^{SK_{\text{Alice}}} \bmod p$
$SK_{\text{Bob}} \in [1, 2, \cdots, q-1]$ — secret key of Bob
PK_{Bob} — public key of Bob where $PK_{\text{Bob}} = g^{SK_{\text{Bob}}} \bmod p$
Operations and functions:
$KH_k(\cdot)$ — a one-way hash function key-ed with k
$hash(\cdot)$ — a one-way hash function
(E_k, D_k) — encryption and decryption algorithms of a private key cipher (attached subscrypts indicate the keys)
$\|$ — Concatenation
$\in_R$ — Random picking (ex.: $x \in_R [1, 2, \cdots, q-1]$ indicates that x is randomly picked from $[1, 2, \cdots, q-1]$.)

Generation of signature (s_1, s_2) on a message m	Signature verification
$x \in_R [1, 2, \cdots, q-1]$ $k = g^x \bmod p$ $s_1 = KH_k(m)$ $s_2 = x/(s_1 + SK_{Alice}) \bmod q$	$k = (PK_{Alice} \cdot g^{s_1})^{s_2} \bmod p$ Accept m if and only if $KH_k(m)$ is identical to s_1.

Fig. 7. Example of SDSS (SDSS1).

Signcryption by Alice		Unsigncryption by Bob
$x \in_R [1, 2, \cdots, q-1]$		$(k_1, k_2) =$
$(k_1, k_2) = hash\,(PK_{Bob}^x \bmod p)$		$hash\left((PK_{Alice} \cdot g^{s_1})^{s_2 \cdot SK_{Bob}} \bmod p\right)$
$c = E_{k_1}(m)$	$(c, s_1, s_2) \Rightarrow$	$m = D_{k_1}(c)$
$s_1 = KH_{k_2}(m)$		Accept m if and only if
$s_2 = x/(s_1 + SK_{Alice}) \bmod q$		$KH_{k_2}(m)$ is identical to s_1.

Fig. 8. Example of signcryption based on SDSS1.