Secure and Lightweight Acknowledgment for Peer-to-Peer Overlay Networks

Lim Chee Liang[*] and Ooi Wei Tsang[†]

Department of Computer Science,
School of Computing,
National University of Singapore

ABSTRACT

We propose a cryptographic protocol for the secure and efficient acknowledgment of peer-to-peer multicast messages. Every node is capable of verifying the acknowledgments upon receiving them, without waiting for and relying on the server. Our protocol is efficient as it is bounded by the maximum depth of the tree, and is resistant to denial-of-service attacks.

1. INTRODUCTION

1.1. Background and Problem

With the rising popularity and importance of peer-to-peer (P2P) networks and multicast, there has been much research into P2P security. However, they mostly focus on authentication and confidentiality of data. Since routing in P2P is performed by untrusted nodes, it can be easily subverted. Furthermore P2P applications typically use unreliable transport such as UDP and thus lack any form of feedback. As a result, it is non-trivial to determine when messages have been received by everyone.

A naive acknowledgment whereby every node replies to the server will easily overwhelm the server with an implosion of messages. The goal is to design an efficient and secure acknowledgment scheme in the context of P2P multicast (or many-to-one transmission protocols in general), which is scalable as the tree grows, and does not require exceptionally high processing power from each node.

1.2. Paper Organization

In Section 1, we give an introduction to the background and problem behind this paper. Section 2 looks at prior work and possible uses of the proposed solution. Section 3 discusses our model and assumptions, followed by a list of notations and definitions in Section 4. Section 5 list the basic concepts behind the solution, followed by discussion of our solution in abstract form in Section 6, and concrete form in Section 7. We then analyze the theoretical performance in Section 8 and simulation results in Section 9. We conclude in Section 10.

2. RELATED WORK AND POTENTIAL USES

Nicolosi and Mazières (2004) proposed an acknowledgment compression scheme for secure and compact acknowledgment of multicast messages. Their protocol relies on a relatively new cryptographic assumption: Gap-Diffie-Hellman (GDH), and is based on the multisignature scheme by Boldyreva (2003). Castelluccia, Jarecki, Kim, and Tsudik (2006) proposed a separate protocol which works on long standing cryptographic assumptions such as discrete logarithm. Their protocols extends existing multisignature schemes (the authors focused on the Schnorr multisignature (Ohta & Okamoto, 1999)), and adds limited robustness using a Merkle's tree.

Proposed for a slightly different purpose, the scheme by Domingo-Ferrer, Martínez-Ballesté, and Sebé (2004) focuses on delivering distinct symbols from every node back to the server, without overloading it with an implosion of incoming connections. They make use of super-increasing sequences in the knapsack problem. However, the final message received by the server is linear to the number of nodes. Thus their scheme does not fit the context of our problem.

Numerous multisignature schemes (Boneh, Gentry, Lynn, & Shacham, 2003; Boldyreva, 2003; Ohta & Okamoto, 1999) exist, which will make acknowledgment aggregation and compression possible. However, robustness in the face of uncooperative nodes is an issue, and these cryptographic schemes typically rely on bilinear maps or other complex cryptographic primitives. Bilinear maps (Dutta, Barua, & Sarkar, 2004) are computationally expensive functions performed over elliptic curve groups.

[*]Student
[†]Supervisor

In the area of IP and P2P multicast, particularly media/video streaming, an acknowledgment scheme can be used to aid streaming authentication schemes which require periodic revealing of secrets. Examples include broadcast MAC protocols like TESLA (Perrig, Canetti, Song, & Tygar, 2001), or one-time-signature schemes like BiBa (Perrig, 2001) and HORS (Reyzin & Reyzin, 2002). Nicolosi and Mazières (2004) mentions other uses such as in confirming the receipt of invalidation messages for cached contents in a content distribution network, or to ensure that security updates are received by every user.

3. MODEL AND ASSUMPTIONS

We assume that the network and P2P multicast overlay may contain an unbounded number of malicious peers, with no bounds on the size of collusion groups. We assume the network to be open to packet sniffing and spoofing. No assumptions are made on the reliability of the underlying routing mechanism (e.g. IP, TCP or UDP). Our scheme's main goal is to be an extension which can be plugged into any existing P2P multicast protocols, hence the acronym ***Slapon***. Thus the protocol assumes the existence of a proper multicast tree.

4. NOTATIONS AND DEFINITIONS

We define **subtree** of node d as the largest subtree in the multicast tree, which is rooted at node d. **Child subtree** refers the subtree of a child node. A node i which has **evidence** of node d implies that i's collection of public keys includes the public key of d. The term **faulty nodes** will be used to encompass both nodes that are not functioning correctly, as well as malicious nodes seeking to disrupt the operations of the multicast tree. A node i is said to **dominate** over another node j, iff i is an ancestor of j in the multicast tree.

For a node i, T_i is the subtree of node i. The secret and public keys of i are s_i and p_i respectively. S_i and P_i are the subtree equivalents of s_i and p_i, and represents the secret and public keys of T_i. For convenience, we also refer to subtree public keys as **subkeys**. We denote the ancestor d-levels above i as i^d, and for a node i with n children, we represent the child nodes as $i_1, \ldots, i_n$. $i_{c(j)}$ denotes the child of i which is dominant over j. **ack**(T_i) denotes he acknowledgment of subtree T_i.

5. BASIC CONCEPTS

The following list of long-standing cryptographic problems and assumptions widely used in cryptographic constructions are employed in the construction of our protocol as well as in proving its security. A more detailed discussion can be found in our full paper (Lim & Ooi, 2007).

- Computational Diffie-Hellman problem.
- Discrete Logarithm problem.
- RSA Problem.
- Hidden Order Modulus Group problem.

All the above problems are assumed to admit no polynomial running-time algorithm capable of solving them, and thus are considered computationally difficult.

6. ABSTRACT MODEL

Rather than relying on the server and waiting for it to check all acknowledgments for errant nodes, we introduce the concept of **incrementally verifiable acknowledgments**. Each node is able to independently verify the acknowledgments from its child subtrees. Consequently, each node is able to deal with faulty child nodes immediately. However, note that our solution is not a replacement for multisignatures. A proper discussion on the reasons is given in our full paper. Proofs of the security of these constructions are also deferred to the full paper.

6.1. Secret and Public Keys

Each node i has a secret key s_i, and the public key is given by $f(g, s_i)$, for some g in the group. Similarly, subtree T_i possesses secret key S_i and public key $f(g, S_i)$. We can combine secret keys through $S_i = G(s_1, \ldots, s_n)$, where $s_i, \ldots\ s_n$ are all the nodes in S_i. It is difficult to invert $f(g, x)$ to obtain x.

6.2. Acknowledgments and Verifications

To request for acknowledgment, the server generates for each acknowledgment round a unique pair of values k_v and k_b, and issues $(f(g, k_v), k_b)$ as the challenge. Node i acknowledge with its own acknowledgment $f(f(g, k_v), s_i)$, combined with the acknowledgments from its child subtrees using an accumulator/aggregator $C()$: $\mathbf{ack}(T_i) = f(f(g, k_v), S_i) = C(\mathbf{ack}(i), \mathbf{ack}(i_1), \ldots, \mathbf{ack}(i_n))$.

To verify acknowledgment from child subtree T_c, the parent verifies that $E(f(f(g, k_v), S_c), k_b) = f(g, S_c)$ $= P_c$, using an extractor function $E()$. In essence, $E()$ compute $E(f(x, k_v), k_b) = x$. To allow this, $f()$ is quasi-commutative: $f(f(g, k_v), S_d) = f(f(g, S_d), k_v)$.

6.3. Joining and Leaving

As noted above, child acknowledgments are computed and verified against the child subkey. Thus every node needs to keep track of their child nodes' subkey. When nodes join or leave, their public keys are accumulated or de-accumulated into ancestor nodes' collection of child subkeys using a function $A()$: $A(P_i, p_c) = P'_i$, where P'_i now contains p_c.

6.4. Efficiency in Size

The individual inputs and final outputs of $C()$, $G()$ and $A()$ are of equal sizes, effectively compressing acknowledgments and keys. Thus our protocol messages are of constant sizes, independent of the multicast tree size.

6.5. Construction Based on Exponentiation and Multiplication

We propose one construction using modular integer arithmetic in an RSA modulus group. Let the keys be elements in $\mathbb{Z}_N, N = pq$. We can then model $f(g, s_i)$ as g^{s_i}. Let $k_v k_b \equiv 1 \pmod{\phi(N)}$, where $\phi(N) = (p-1)(q-1)$. Acknowledgments are then computed as $f(f(g, k_v), s_i) = g^{k_v s_i}$. Since $f()$ is now quasi-commutative, i.e. $(g^{k_v})^{s_i} = (g^{s_i})^{k_v}$, let $E(f(x, y), z) = f(x, y)^z$. Verification can then be performed as $E(f(f(g, k_v), s_i), k_b) = (g^{k_v s_i})^{k_b} = (g^{s_i})^{k_v k_b} = g^{s_i}$, fulfilling the requirements of the abstract model above.

Another possible construction uses elliptic curves to obtain smaller key sizes. Details of this as well as proofs of security for the constructs in this section can be found in the full paper.

7. PROTOCOL

Next we describe the actual protocol as applied to a P2P multicast application.

7.1. Initial Setup

The server creates a new session by generating parameters of the group to operate in. For a group $\mathbb{Z}_N$, the parameter are N, p, q in $N = pq$.

7.2. Joins

A nodes i joining the tree obtains the group parameters, secret and public keys from the server. It attached to a parent node j, obtains the parent's public key p_j and notifies the server. i also agrees on a shared secret key with j. The server notifies every ancestor node of i of its entrance, and provides the new node's public key for accumulation.

To authenticate the join messages, the server provides join message of new node i to ancestor node a in the form $(p_i, h_{s_a + p_c}(p_i))$, where c is the child node of a which is dominant over i, and h_k is a keyed hash with input key k. Ancestor nodes can thus determine which child subkey to accumulate into by calculating the hash for every child node, and can be assured that the join message is indeed intended for it, without using expensive digital signatures.

7.3. Drops and Departures

The parent node informs server of the drop or departure of a child node. The server sends all ancestor nodes a drop message which instructs them to de-accumulate the dropped node's public key. The format of drop messages is similar to that of join messages, where a hash/digest component provides authenticity at a low cost. As a stop-gap measure to allow acknowledgment to carry on without waiting for ancestors to de-accumulate the public key, the server may issue the parent node with the dropped subtree's acknowledgment, allowing it to forward a valid acknowledgment up the multicast tree.

	Node	Each ancestor node		Server	
		HashComp	BigNumOp	HashComp	BigNumOp
Joins	$O(1)$	$O(c)$	$O(1)$	$O(d)$	$O(d)$
Drops & departures	$O(1)$	$O(c^2)$	$O(c)$	$O(dc)$	$O(dc)$

	Node (including server)	
	SymmCiph	BigNumOp
Verifications & acknowledgments	$O(c)$	$O(c)$

Table 1: Computation overhead for protocol. d is the maximum depth of the multicast tree, and c is the maximum out-degree. HashComp refers to hash operations, BigNumOp refers to large number arithmetic and SymmCiph refers to symmetric encryption and/or decryption.

7.4. Handling Faulty Nodes

Nodes which fail to forward acknowledgments or forward invalid acknowledgments are considered faulty and dropped immediately by the parent. *Proof:* if child c of node i is not faulty, then a node below it is faulty. If this is true, the parent of the faulty node would have dropped it and the server would have issued the drop messages or stop-gap acknowledgments. Thus i should not receive invalid acknowledgments.

7.5. Rejoins

When a node d is dropped, all the nodes below it are affected. To prevent an implosion of rejoins from all these nodes, the server will immediately rejoin the child nodes of d. Rejoin messages are sent to the new ancestors of the rejoined nodes to update them of the rejoined nodes' subtree keys, in a manner similar to join messages.

7.6. Acknowledgments

Nodes issue acknowledgment in response to an acknowledgment challenge from the server. They acknowledge only if they have successfully received the multicast data for the current round. The acknowledgment is a combination of their own acknowledgment, computed with $f(f(g, k_v), s_i)$, and their child nodes' acknowledgments, using the accumulating/aggregating function $C()$.

7.7. Preventing Replay and Denial-of-service Attacks

All join, drop and rejoin messages should contain increasing sequence numbers to guard against replay attacks. To guard nodes against spoofed acknowledgment messages and a flood of such messages, which will trigger expensive exponentiation operations, we propose the use of symmetric encryption. Acknowledgment messages are encrypted using the shared key between a child and parent node, together with a unique identifier of the child. Thus messages which do not decrypt to obtain the child's identifier can be discarded.

8. PERFORMANCE ANALYSIS

Performance bounds. In terms of computational power and communication bandwidth required, our protocol has an upper bound of the maximum depth of the multicast tree, which in a well distributed multicast tree is typically close to $\log n$, where n is the number of nodes in the tree. Tables 1 and 2 summarizes the computational complexity and communications overhead for the protocol components. A detailed analysis can be found in our full paper.

Performance under denial-of-service. Both join and drop operations require only hash operations to verify the messages before performing the more expensive large number exponentiation to accumulate/de-accumulate the key. Thus a large number of invalid messages will not result in a denial-of-service. During the verification phase, we must perform one exponentiation per acknowledgment from a child node, provided the acknowledgment decrypts and verifies correctly via the shared symmetric cipher key between parent and child node. Therefore a flood of malicious acknowledgment messages will only result in less costly symmetric decryption and not large number exponentiation.

Optimization. Based on the performance analysis, we observe and propose an optimization: pre-generation of public and secret keys for new nodes. Since the generation of keys is the only process other than acknowledgments and verifications that requires exponentiation, we can generate keys and cache them during periods of relatively lower CPU usage, providing some resistance against computation spikes when the rate of joins and drops increases.

In conclusion, we expect our protocol to perform well even in large multicast trees and under the effect of hostile nodes and external parties.

	Node	**Each child node**	**Each ancestor node**	**Server**		**Node (incl. server)**
Joins	$O(1)$	n/a	$O(1)$	$O(d)$	Verifications & acknowledgments	$O(c)$
Drops & departures	n/a	$O(1)$	$O(c)$	$O(cd)$		

Table 2: Communication overhead for protocol. d is the maximum depth of the multicast tree, and c is the maximum out-degree.

512 bits key			**1024 bits key**		
Injected events/s	Delay (s)	Total events/s (incl. rejoins)	Injected events/s	Delay (s)	Total events/s (incl. rejoins)
10	0.0722	16.2	10.4	0.0776	15.6
24.8	0.148	33.2	30.6	0.193	43.2
30.6	0.114	43.8	50.2	0.237	80
51.8	0.189	73.6	62.2	0.159	86.2
62.2	0.159	86.2	80.8	0.376	127.6
101.6	0.300	138	91.2	0.401	129.5
110	overflow		100	overflow	

Table 3: Simulation results for 512 and 1024 bits keys. Total events per second includes rejoins, which results from drops.

9. SIMULATION

In order to provide a sense as to the practicality of our proposed protocol, we implemented a library to provide the necessary functions of the protocol, and created a simulator to measure its performance.

9.1. Cost of Large Number Operations

Asymmetric cryptography often involve the use of large number arithmetic. However, due to their high computational cost, multicast authentication often either shy away from them, or attempt to amortize the cost over multiple packets. To give a rough sense of the state of things, we performed a simple measurement of large number operations.

On a Pentium-M 1.86 GHz notebook computer, we measured 127020 multiplications per second versus 225 exponentiations per second, for 512 bits numbers. For 1024 bits numbers, observed 71334 multiplications versus 42 exponentiations per second[3]. We observe that the bottleneck lies in exponentiation operations, which are around 550 and 1700 times slower than multiplications for 512 bits and 1024 bits values respectively.

9.2. Simulation of Protocol

Given the observation that the most variable component of the protocol, in terms of resources required, is joins and drops at the server, particularly under high churn rate in the multicast tree, the simulation focuses on server-side performance under varying number of joins, drops and the resulting rejoins.

New nodes are randomly attached to a parent to simulate a well distributed tree. The maximum out-degree is 5. Events are injected when the tree reaches a size of 1000 nodes. The server simulator was ran on a Pentium-4 3.0 GHz desktop computer, and the node tree simulator was ran on a Pentium-M 1.86 GHz notebook computer. Table 3 document the results of the simulations. Delay values refer to the time elapsed between receivers sending out a message and server processing it.

9.2.1. Analysis of Results

For the key size of 512 bits, we can observe that when the server receives more than 100 join and drop messages per second, which results in more than 138 join, drop and rejoin events per second, the server is no longer able to keep up. For the key sizes of 1024 bits, this maximum point occurs at 90 events per second.

Of notable curiosity is that key sizes of 1024 bits did not double the computation workload, resulting in the simulation reaching the maximum point at half the number of events as compared to key sizes of 512 bits. This highlights the importance of other factors such as large number multiplications, hashing and operations in maintaining records for every node.

In a practical context of a real multicast application of around 1000 peers, if we assume that the server must devote 50% of its processing capacity to authentication, encoding, streaming and other system functions, then it should be able to handle up to an average of 45 joins and drops per second. In a less

[3]The results were obtained using C++ code and the GNU Multiple Precision (GMP) arithmetic library.

optimistic setting where the server must devote 75% of its processing power to other functions, it can manage around 20 joins and drops per second.

We conclude that the proposed protocol, despite requiring expensive large number operations, has acceptable performance even in large multicast trees. Chou, Padmanabhan, and Wang (2003) notes that for a flash crowd accessing a live streaming news video on September 11, 2001, the average rate of joins and drops was 180 per second. The peak rate was 1000 per second. This further validates the practicality of our protocol in a typical, less hostile environment.

10. CONCLUSION

We have achieved, through the proposed protocol, a secure and lightweight method for performing acknowledgments of messages in a peer-to-peer multicast network. Our protocol achieves efficiency, wherein all components of the protocol have an asymptotic communication and computation upper bound of the maximum depth of the multicast tree. This provides a very optimistic bound on our protocol, and provides scalability even in huge P2P multicast trees.

Through the use of less costly symmetric cryptographic operations such as MAC and symmetric ciphers, we are able to create resistance against denial-of-service attacks and spoofing of network packet source. Compared to other protocols, our protocol has made the trade-off of requiring the server to maintain the state of every node in the tree, and requiring every ancestor node to be notified of every joins and drops. In return, we obtain the ability to perform incremental verification of acknowledgments.

11. REFERENCES

[1] Boldyreva, A. (2003). Efficient treshold signature, multisignature and blind signature schemes based on the Gap-Diffie-Hellman-group signature scheme. *Public Key Cryptography - 6th International Workshop on Theory and Practice in Public Key Cryptography (PKC 2003)*, Miami, FL, January, 2003.

[2] Boneh, D., Gentry, C., Lynn, B., & Shacham, H. (2003). Aggregate and verifiably encrypted signatures from bilinear maps. *Eurocrypt 2003*, Warsaw, Poland, May, 2003.

[3] Castelluccia, C., Jarecki, S., Kim, J., & Tsudik, G. (2006). Secure acknowledgment aggregation and multisignatures with limited robustness. *Computer Networks*, *50*(10), July, 2006, 1639–1652.

[4] Chou, P., Padmanabhan, V., & Wang, H. (2003). Resilient peer-to-peer streaming. *11th IEEE International Conference on Network Protocols (ICNP2003)*, Atlanta, GA, November, 2003.

[5] Domingo-Ferrer, J., Martínez-Ballesté, A., & Sebé, F. (2004). Secure reverse communication in a multicast tree. *3rd International IFIP-TC6 Networking Conference (Networking 2004)*, Athens, Greece, May, 2004.

[6] Dutta, R., Barua, R., & Sarkar, P. (2004). Pairing-based cryptographic protocols : A survey. Available at: http://eprint.iacr.org/2004/064.

[7] Lim, C. L., & Ooi, W. T. (2007). *Secure and lightweight acknowledgments for peer-to-peer overlay networks.* UROP report.

[8] Nicolosi, A., & Mazières (2004). Secure acknowledgement of multicast messages in open peer-to-peer networks. *3rd International Workshop on Peer-to-Peer Systems (IPTPS 2004)*, LA Jolla, CA, February, 2004.

[9] Ohta, K., & Okamoto, T. (1999). Multi-signature schemes secure against active insider attacks. In *IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E82-A (pp. 21–31).

[10] Perrig, A. (2001). The BiBa one-time signature and broadcast authentication protocol. *8th ACM conference on Computer and Communications Security*, Philadelphia, PA, November, 2001.

[11] Perrig, A., Canetti, R., Song, D., & Tygar, J. D. (2001). Efficient and secure source authentication for multicast. *Network and Distributed System Security Symposium 2001 (NDSS 01)*, San Diego, California, February, 2001.

[12] Reyzin, L., & Reyzin, N. (2002). Better than BiBa: Short one-time signatures with fast signing and verifying. *Information Security and Privacy: 7th Australian Conference (ACISP 2002)*, Melbourne, Australia, July, 2002.