Distributed Authentication for Peer-to-Peer Networks

Shardul Gokhale and Partha Dasgupta
Arizona State University
Tempe, AZ 85287-5406
{shardul, partha}@asu.edu

Abstract

A public key infrastructure is generally (and effectively) used for cryptographically secure authentication in the networks. Ad-hoc networks are formed in haphazard manner. Security services for ad-hoc networks cannot assume the existence of a particular infrastructure. Peer-to-peer technology is promising in addressing security issues in ad-hoc networks. We provide a novel; cryptographically secure representation of trust based on secure groups - troups. We show how troups can be constructed in a distributed manner using RSA accumulators. The troup-membership is verified using the zero-knowledge protocol of modular exponentiation. Each node in a group has an identity within a group, but it is not required to reveal the identity during verification. This trust model is not centrally controlled and can be deployed incrementally in the network. This paper presents the protocols and a prototype implementation of the troups based authentication system.

1 Introduction

Ad hoc networks are formed in a "haphazard" manner and do not rely on an established infrastructure. These physical properties of ad-hoc networks are similar to peer-to-peer networks at logical level. We feel that peer-to-peer techniques can be used on ad-hoc networks to enhance efficient usage of the underlying topology. Security (i.e. authentication and trust management) is based on the notion of trust and it is not well defined for ad-hoc networks. Centralized security solutions for ad-hoc networks could be significantly vulnerable [2].

Recently, a lot of effort has been put in providing security infrastructure for ad hoc networks [2, 3, 4, 5]. These efforts aim at distributing CA functionality to a set of nodes in the network. In this paper, we propose, *troups*, a cryptographically secure, fault-tolerant, scalable trust model for resilient peer-to-peer networks. Our design employs a secure group based approach for the creation of a distributed trust model. In our opinion, the pure peer-to-peer organization and the flexibility provided by this model make it suitable for ad hoc networks. During the formation of a troup a node generates an identity within a troup. However, it may choose not to reveal the identity during verification. These troups are dynamic in nature and support the addition and deletion of nodes as well as the troup-merge operations. Overlapping troups can form a network-wide security infrastructure.

The rest of the paper is organized as follows: Section 2 provides the background and motivation for the development of such a trust model. The core idea of the trust model based on a secure group construct is illustrated in Section 3. Section 4 provides the security protocols for this trust model. It also illustrates that the constructs are cryptographically secure. The implementation of the verification protocol and performance results are provided in Section 5.

2 Background

Recent publications [2, 4, 5] propose the idea of making existing authentication models in the Internet (Hierarchical and PGP) [1] suitable for mobile ad hoc networks.

2.1 Threshold Cryptography

In [2, 4] the use of threshold cryptography is proposed to distribute CA functionality to various nodes. In [4] private key of certifying authority is shared between a few nodes. These nodes can sign

a certificate using the respective shares. The node, for which this certificate is generated, collects these shares and combines them to generate a certificate. A centralized trusted node is needed to initialize the shares. Due to the support provided for generating additional shares, a compromised node can steal the CA's private key if has k shares. This problem is significant because it is difficult to revoke the CA's signing key.

2.2 Certificate Chains

The approach proposed in [5] is based on the PGP authentication system. In this pure peer-to-peer approach a node updates a sub-graph of the certification graph of the network periodically. Whenever two nodes want to authenticate each other, sub-graphs are merged with or without helper nodes in an attempt to create a certificate chain. If the certificate chain exists, the node is authenticated. Due to dynamic nature of ad hoc networks, revocations play an important role.

2.3 Security and Cost

No network can be 100% secure. The security of the network is trade-off between the security needs and the cost associated with it. The aforementioned security models provide proactive approach, so revocation is a costly operation. A network that is easier to compromise can be made resilient to such attacks. As ad hoc networks are dynamic and formed without infrastructure, a resilient approach may be better for security.

3 New Trust Model

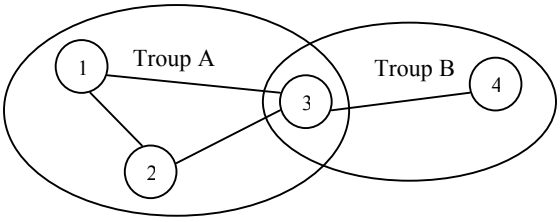

Fig. 1. A Trust Model based on Troups

In our model, trust is represented by a group membership. Whenever two nodes in a network decide to establish a trust relationship, a new group, which consists of the two nodes, is formed. We call such trust-based groups, *troups*. In this paper, we propose a new protocol for troup formation - creation of RSA accumulator in a distributed manner. To support the dynamic membership in a troup, we have developed troup mutation protocols. These protocols are based on CLIQUES protocols. Membership verification is based on the zero-knowledge protocol for modular exponentiation. Thus, even a non-member can verify troup membership.

This representation of trust also supports transitive trust. For the network shown in Fig. 1, an immediate trust relationship exists between nodes 1 and 3, and nodes 3 and 4. Node 1 can verify the transitive trust that it has with node 4 by verifying the immediate trust between itself and node 3, and nodes 3 and 4. Transitive trust is helpful in establishing chains of trust in a network. Troups also provide bi-directional trust representation. If a member within a troup starts behaving maliciously, the other troup members can revoke the trust simply by removing the member from the troup.

4 Tools

We use Collision-Free One Way Accumulators [9,10] for the construction of troups. To verify membership, we use zero-knowledge proof for modular exponentiation [11]. In [9], Benaloh and de Mare, propose a cryptographically secure construct – One-way Accumulator (OWA). The same construct is used for generation of troups. Each to-be-member generates an exponent, y_i - that is to be accumulated. They also agree on the seed of the accumulator, x, and modulus, N.

Accumulation $z = x^{y_i * y_2 * \cdots * y_n} \mod N$

and auxiliary value

$aux_i = x^{y_1 * y_2 * \cdots * y_{i-1} * y_{i+1} * \cdots * y_n} \mod N$

The accumulation (troup identifier) is public, - used to represent a group. As the membership of an accumulator cannot be forged, the accumulation becomes the public key of the group. To verify the membership of troup (z), the member can present the values (aux_i, y_i) to the verifier. The membership is verified if $z = aux_i^{y_i} \bmod N$. However, if the values

(aux_i, y_i) are revealed to the verifier it can be used maliciously. The other approach would be to prove the knowledge of the two values (aux_i, y_i) such that z can be constructed by modular exponentiation. The zero-knowledge protocol for modular exponentiation proposed in [11] by Camenisch and Michels can be used for this purpose.

4.1 Troups Formation and Membership Verification

In this section, we propose secure generation of accumulator in a distributed manner. z is the accumulation and N is RSA modulus. The accumulator is based on an elementary accumulator function i.e. $x^y \bmod N$. Security of the troup is considered only against passive attackers. Just for the illustration, assume a central computing authority (CCA) and an accumulator seed, x. In actual troups implementation a designated troup controller replaces CCA and accumulator seed is constructed in a contributive manner. A regular RSA accumulator is generated as follows:

1. Each participant generates a large prime number, y_i.
2. Participants send respective y_i to the CCA.
3. CCA calculates auxiliary values, aux_i, corresponding to, y_i.
4. CCA transmits aux_i appropriately.

To prevent passive attacker from listening (aux_i, y_i) values during accumulator construction, the communication can be encrypted with a secret key. The extra round needed for encryption can be avoided by slight modification in the protocol. The strongly one-way property of the accumulator is defined in [11] as *"...given only* $(y_1, \ldots, y_N)$ *and* z*, it is hard to find a pair* $(y', accu')$ *such that authentic*$(z, y', accu')$ *= ok with* $y' \notin \{y_1, \ldots, y_N\}$*."* This property suggests that an accumulator is not compromised even if the attacker can choose the values of y'. To ensure that the attacker does not choose the values y_i, used during the accumulator construction; the verification function, *authentic* $(z, y', accu')$*,* is changed in order to accept only a particular set of y'. We propose the following technique for RSA accumulator construction:

1. Each participant generates two large prime numbers, p_{1i}, p_{2i} and product $y_i = p_{1i} * p_{2i}$.
2. Participants send respective y_i to the CCA.
3. CCA calculates temporary auxiliary values, $taux_i$, that correspond to, y_i.
   $taux_i = x^{y_1 * y_2 * \ldots * y_{i-1} * y_{i+1} * \ldots * y_n} \bmod N$
4. CCA transmits $taux_i$ to respective participants.
5. The participants then calculate $aux_i = taux_i^{p_{2i}} \bmod N$.

Thus, each participant effectively generates two members. The attacker can snoop on y_i values but can never know p_{1i} and p_{2i}. To ensure that $(taux_i, y_i)$ are not used during membership verification, a verification function is also made to check for the bit-length of the exponent. If the prover exceeds the restricted bit-length of the exponent, the verification protocol terminates with rejection. The restricted bit-length is such that bit-length (y_i) > restricted bit-length > bit-length (p_{1i} or p_{2i}). Protocol proposed in [11] satisfies this requirement.

The security of troups trust network depends on the security of all the overlapping groups in the network. The strongly one-way property of the accumulator ensures that finding $(y', accu')$ for a specific troup is hard. Security of troups network may be compromised if it is easy to find collisions. The RSA Accumulator [10] makes troups secure against collisions. Under strong RSA assumption, collision-free property of RSA accumulator ensures that finding $(y', accu')$ for any group in troups trust network is hard to find.

The seed of the accumulator can be generated in a secure manner using a multiparty diffie-hellman key agreement protocol. In this paper, we present specific techniques based on CLIQUES [7] for troups. The advantage of using CLIQUES for RSA accumulator generation is that the communication for accumulator construction can be integrated with the CLIQUES key agreement

protocol. In general, any technique that confirms to the above requirements e.g. STR, TGDH [13] can be used to create of troups.

Protocol for troup formation:

The troup formation protocol shares the communication for accumulator generation with the IKA.1 protocol of CLIQUES; and hence it takes n rounds for completion. First n-1 rounds include collection of exponents from all the participants. In the last round (n), the last participant receives all the exponents. As this round results in the completion of the IKA CLIQUES protocol, the last participant knows the seed, x, and the exponents ($y_1, y_2 \ldots, y_n$). It then broadcasts, $taux_i = x^{y_1 * y_2 * \ldots * y_{i-1} * y_{i+1} * \ldots * y_n} \bmod N$, corresponding to i^{th} member. The members calculate respective auxiliary values, aux_i, and accumulation, z.

For the first n-1 members:

1. M_i receives a set of i-1 exponents from first M_{i-1} members.
2. M_i generates two prime numbers , p_{1i} and p_{2i}. The value $y_i = p_{1i} * p_{2i}$ is added to the set of exponents.

The highest indexed participant, M_n acts as the troup controller, which calculates intermediate auxiliary values for all the members and broadcasts them to the members.

1. M_n calculates the seed, x, in the last round of IKA protocol.
2. The intermediate auxiliary value $taux_i = x^{y_1 * y_2 * \ldots * y_{i-1} * y_{i+1} * \ldots * y_n} \bmod N$, which corresponds to i^{th} member, is calculated. M_n broadcasts these values to the members.

Each member needs to find an auxiliary value for membership proof. They proceed as follows.

1. calculate $aux_i = taux_i^{p_{2i}} \bmod N$. ($aux_i$, p_{1i}) becomes the primary key of troup member.
2. The accumulation z is also calculated. z is the public identifier of the troup.

Protocol for membership verification:

The protocol used for membership verification is the zero-knowledge protocol for modular exponentiation. A detailed design of the protocol can be found in [11], but we are providing short description for completeness. The protocol provided in [11] hides all the numbers that are used in modular exponentiation. E.g. To prove knowledge of a, b, d, and n, such that $a^b \equiv d \bmod n$, the prover generates commitments for all the numbers and does not reveal the actual numbers. In the case of troups d and n are already known to the verifier. This does not pose a threat to the security of zero-knowledge protocol. Such an approach is already used in [12].

The verifier and prover start off by agreeing on a commitment scheme for the protocol. The prover sends commitments for aux_i, p_{1i}, z, n, and the intermediate results of square and multiply algorithm. Commitments for z and n are opened by the prover. The prover then sends the responses to verifier for the challenges sent. If the verifier can reconstruct commitment for z from the initial commitments the verification succeeds. Details about implementation of this protocol and performance results are provided in section 6.

4.2 Troups Mutation

Troup membership changes throughout its lifetime. To facilitate troup changes we have retained the concept of group controller from CLIQUES. Protocols for troup mutation are based on CLIQUES AKA protocols. Troup mutation relies on the ability of AKA protocols to generate the new seed value. The specific communication with the troup members and rights to add/delete a member depend on troup policy. In general, any troup member that retains the contribution of earlier participants can be a troup controller and has ability to mutate a troup.

Member Addition

To join a troup, the participant provides a new seed and an accumulator contribution. It is assumed that the incoming member becomes the troup controller. The addition of a member can be considered as an extension of the last step of the troup genesis.

Member Addition protocol:

1. The new group controller generates its contribution for the seed and accumulator y_{n+1}.
2. The old troup controller sends the contents of the last broadcast message to the new troup controller. This message includes the set of exponents contributed by all the troup members for generation of accumulator i.e. $(y_1, \ldots, y_N)$
3. The new controller generates the intermediate values required for calculation of the auxiliary values.

Mass Join and Group Fusion:

It may sometimes be necessary to add multiple members to the troup. It can be done by chaining the members to be added as in the case of CLIQUES. The new group controller then calculates the intermediate values for accumulator and broadcasts those values to the members. A special case of mass join is group fusion. In this case, the incoming members have pre-established relationship. These relationships can be used to calculate new seed as in CLIQUES.

Member Exclusion:

To remove a member from a troup, the troup controller generates a new seed using earlier contribution (as specified in the AKA). This new seed is used for accumulation of values. As the outgoing member cannot differentiate the new seed from any random number, the security of a troup is guaranteed. Multiple members can also be removed using this strategy. In fact it is easier to remove multiple members - in the sense that, fewer numbers of intermediate values need to be generated.

4.3 Member Identification

To verify, the membership of a particular group, a member is not required to identify. However, if desired, a member could be made to reveal the identity within a troup. The product y_i provided by the node during the construction of an accumulator, becomes the public identifier of the node. To verify that a node is a member, it needs to prove following two statements in zero-knowledge.

- The factors - p_{1i} and p_{2i} - of the public identifier y_i are known to the prover.
- One of the factors p_{1i} is used in the verification of membership. i.e. $z = aux_i^{\,p_{1i}} \bmod n$.

The commitment of a number x will be generated as $C_x = g^x h^r \bmod n'$ where (g, h, and n') is the commitment scheme used for the protocol [11].

The protocol is executed in the following manner.

1. The prover generates commitments C_z, C_{aux_i}, C_{p1_i} and C_n which correspond to z, aux_i, p_{1i}, and n.
2. The verifier proves modular exponentiation by using the zero-knowledge proof for modular exponentiation.
3. The verifier proves the representation of g^{y_i} to the bases C_{p1_i} and h, i.e. $g^y \equiv C_{p1_i} h^v$

5 Implementation and Performance

The protocol for membership verification is implemented in C++ using Crypto++ API for cryptographic functions. The verification protocol is between two agents, viz. ProverAgent and VerifyAgent. The ProverAgent proves knowledge of (aux_i, p_{1i}) such that on modular exponentiation in mod n, troup identifier z is generated.

This stateful protocol is divided into the following five states.

Initialization State (T_1): In this state, a secure environment for the execution of zero-knowledge protocol is generated. Specifically, a commitment scheme for the verification protocol is generated. The commitment scheme implemented in this protocol is same as the one used in [11]. The ProverAgent generates a group of appropriate order, N, generators, g and h, and sends it to the VerifyAgent. The ProverAgent uses the

commitment scheme - g, h, and N for generating commitments.

Commitment State (T_2): The ProverAgent generates various commitments. The commitments C_z, C_{aux_i}, C_{p1_i} and C_n correspond to z, aux_i, p_{1i}, and n. It also generates commitments for individual bits of exponent p_{1i}, modular powers of aux_i, and intermediate values of square and multiply algorithm. These commitments are saved by the ProverAgent and sent to the VerifyAgent at the end of this state. The ProverAgent uses these commitments for generating responses and VerifyAgent uses them for verification.

Challenge State (T_3): In this state, the VerifyAgent saves the commitments sent by ProveAgent. The VerifyAgent randomly generates the challenges. It saves the challenges, as they are required during verification and sends them to the ProverAgent.

Response State (T_4): Depending upon the challenges sent by the VerifyAgent and initial commitments, the ProverAgent calculates the responses. It sends the responses to the VerifyAgent and deletes the state for the current protocol.

Verification State (T_5) The VerifyAgent verifies the responses. If all the responses are proper according to the zero-knowledge proof, the verifier accepts the membership.

For calculating the performance we measured $T_4 - T_1$ and $T_5 - T_1$ as the time consumed by ProverAgent and VerifyAgent resp. Actually, the ProverAgent is idle for time period $T_3 - T_2$ and the VerifyAgent is actually idle for $T_4 - T_3 + T_2 - T_1$. In practical implementation time can be multiplexed.

The performance of verification protocol mainly depends on the following factors:

1. Bitlength of secret values (aux_i, p_{1i}):
2. Bitlength of commitment scheme (g, h, n):

The performance measurements are done on windows 2000 PCs running on P3 processors. Both the machines were lightly loaded at the time of performance measurements. We measured the performance of the protocol by changing the bitlength of the exponents as well as commitment scheme.

Commitment scheme is set at 128-bit:

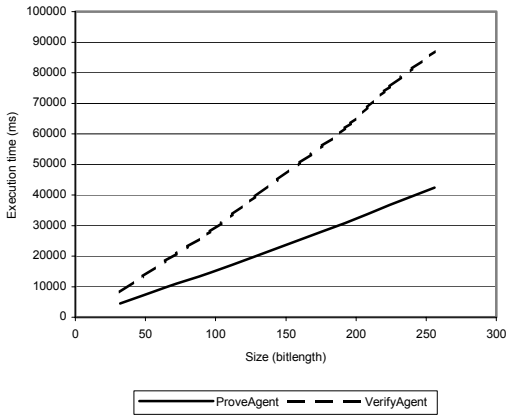

Fig. 2 Size of secret values changed

The bitlength of secret values is set at 128-bit:

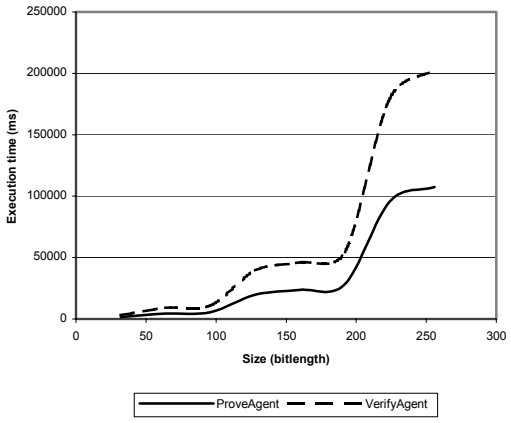

Fig. 3 Size of commitment scheme changed

The bit-length of both the values varies from 32-bit to 256-bit.

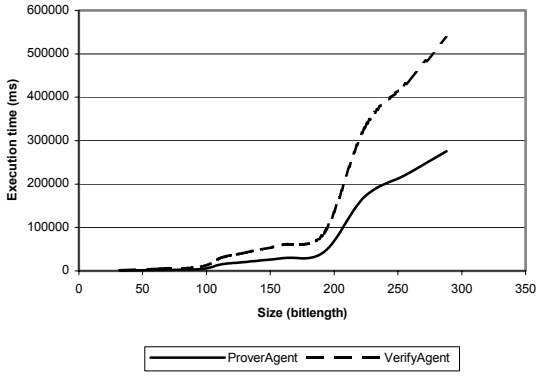

Fig 4 Size of commitment scheme and secret values changed

As it can be seen from Fig. 2, execution time of the verification protocol increases linearly with the increase in the bit-length of the secret values. Fig. 3 illustrates that the execution time rises sharply when commitment scheme is 128-bit and 224-bit. The increase for in-between intervals is not so sharp. Graph in fig. 4 is combination of both the changes. It can be concluded from the above experimental results do not look promising enough to implement the protocol in low-end mobile devices. The sharp rise in curves at 128-bit and 224-bit commitment scheme seems specific to implementation of modular exponentiation. Support from optimized hardware and software specific to this functionality will be needed for practical implementation of this protocol.

6 Conclusion

We have presented a novel way of representing trust relations between the nodes in the network using secure groups construct, *troups*. We have shown how *troups* can be created in a pure peer-to-peer manner. We have suggested protocols for handling changes in troups membership. Each node has an identity inside a troup. A node may choose not to reveal its identity during the verification process, maintaining anonymity in the network. A network of nodes can thus be secured incrementally and distributed manner using these constructs. The mobility of the nodes is handled by localizing the impact of trust revocation. The verification protocol is based on the zero-knowledge protocol for modular exponentiation. We have implemented the protocol to test its performance.

References

[1] Adams, C. M., Mike Burmester, Yvo Desmedt, M. K. Reiter, and Philip Zimmermann, "Which PKI (public key infrastructure) is the Right One? (Panel session)." ACM Conference on Computer and Communications Security 2000: 98-101.

[2] Zhou, L., and Z. J. Haas "Securing Ad Hoc Networks", IEEE Network, 1999.

[3] Weimerskirch, André, and Gilles Thonet "A Distributed Light-Weight Authentication Model for Ad-hoc Networks," v. 2288 of LNCS, 2002.

[4] Luo, Zerfos, Kong, Lu and Zhang, "Self-securing Ad Hoc Wireless Networks," ISCC '02

[5] Botelho, Luis, Steven Willmott, Tianning Zhang and Jonathan Dale, "Self Organized Public Key Management For Mobile Ad Hoc Networks," Technical Reports in Computer and Communication Sciences, 2002.

[6] Abdul-Rahman, A. and Stephen Hailes, "A Distributed Trust Model," In proceedings of the 1997 New Security Paradigms Workshop, ACM Press, 1998: 48-60.

[7] Steiner, Michael, Gene Tsudik, and Micheal Waidner, "CLIQUES: A New Approach to Group Key Agreement." IEEE Transactions on Parallel and Distributed Systems, August 2000.

[8] Ateniese, Giuseppe, Michael Steiner and Gene Tsudik, "New Multi-party Authentication Services and Key Agreement Protocols." IEEE Journal of Selected Areas in Communications, April 2000.

[9] Benaloh, Josh, and Michael de Mare, "One Way Accumulators: A Decentralized Alternative to Digital Signatures," Advances in Cryptology – Proceedings of Eurocrypt '93, Springer-Verlag, 1993.

[10] Baric, Niko, and Birgit Pfitzmann, "Collision-free Accumulators and Fail-stop Signature Schemes without Trees" Advances in Cryptology – Eurocrypt '97, v. 1233 of LNCS, Springer-Verlag, 1997: 480-494.

[11] Camenisch, Jan, and Markus Michels, "Proving in Zero-knowledge that a Number is the Product of Two Safe Primes." Advances in Cryptology – Eurocrypt 1999, v.1592 of LNCS, Springer – Verlag, 1999: 106-121.

[12] Sander, Tomas, Amnon Ta-Shma and Moti Yung. "Blind, Auditable Membership Proofs," Financial Cryptography 2000: 53-71.

[13] Kim, Yongdae, Adrian Perrig and Gene Tsudik, "Simple and Fault-Tolerant Key Agreement for Dynamic Collaborative Groups." 7th ACM conference on Computer and Communication Security, November 2000.