Authentication Under Denial-of-Service Attacks

BETTINA WEISS
Technische Universität Wien
Department of Automation
Treitlstraße 1, A-1040 Vienna
bw@auto.tuwien.ac.at

ABSTRACT
This paper is concerned with authenticated communication using public keys, which has been designed to minimize the potential for denial-of-service attacks in an ad-hoc network. To this aim, we use substantial hardware support and require certain properties from the network. We show how communication partners acquire public keys and how keys can be revoked and changed. We consider various attacks and analyze their effect on our system. It turns out that we can diminish the attack possibilities, but cannot completely protect our system from denial-of-service attacks due to its ad-hoc nature.

KEY WORDS
Authenticated Communication, Denial-of-Service Attacks, Client Puzzles, Message Authentication Codes, Ad-hoc Networks, Hardware Message Filter

1 Motivation

Over the last decades, security has become an increasingly important issue in the design of distributed systems. One vital aspect of security is authentication, which allows a node to identify its communication partners. Apart from intrusion prevention, fundamental algorithms for distributed systems like e.g. consensus algorithms [1] also benefit from authentication.

The challenge of developing security algorithms is to ensure that their security goals are never breached even in the presence of malicious intruders. But it is equally important to provide resistance against denial-of-service (DoS) attacks on the protocol, which cause a node to respond more slowly or not at all to legitimate requests by consuming resources through faked requests. DoS attacks may also suppress valid messages and thereby cause nodes to wait unduly long. Since our W2F architecture[1] is concerned about the quality of its services and fast response times, we are particularly interested in mechanisms that prevent DoS attacks as much as possible.

Given the importance of the problem, it has already been investigated from different angles in the literature. Zhou and Haas [2] also consider security in ad-hoc networks. Their goal is to secure the routing service from attacks by using signatures and by relying on connection redundancy. For them, denial-of-service means that wrong routing information is injected; they do not consider a slow-down of the service.

Matsura and Imai [3] provide strategies for combating DoS attacks. Their basic approach is to employ a "falling-together" strategy where the attacker has to pay a computational cost that is comparable to that of the target. The attacker should also make the first move, i.e., be the first one to employ heavy computation which can easily be verified. Only after the attacker has shown its commitment by performing the required computations should the target do any heavy computations on its own part. The same approach is proposed by Aura et al. in [4], where the authors use client puzzles [5] to make the client commit its resources first. An appealing feature of the approach is the ability of the server to choose the difficulty of the puzzle by a parameter sent with the challenge. This allows a server to react to increasing overload conditions by dynamically increasing the difficulty of the puzzles.

Meadows [6] proposes a framework for the analysis of denial-of-service attacks. It allows the protocol designer to compute the costs of such an attack to the defender and to compare it with the costs to the attacker.

In our paper, we extend the client puzzle approach of [4] to peer-to-peer communication and integrate message filtering at the hardware level to diminish the effects of DoS attacks. The remainder of the paper is organized as follows: We will first present our system model in Section 2 and state our requirements on the hardware in Section 3. Section 4 will present our authentication algorithm and informally analyze its resilience to DoS attacks. Sections 5 and 6 are dedicated to key publication and key revocation, respectively. In Section 7, we consider what happens if our assumptions are violated. Section 8 concludes the paper.

2 System Model

We assume a set of n nodes interconnected by some ad-hoc network, i.e., a network whose topology is unknown. Nodes are either correct or *intruders*, with at most f_i intruders in the system. Any two nodes are connected ei-

[1]This research is part of our W2F-project, which targets a wire-line/wireless fieldbus based upon spread-spectrum (CDMA) communications, see *http://www.auto.tuwien.ac.at/Projects/W2F/* for details. W2F is supported by the Austrian START programme Y41-MAT.

ther directly or over at least $f_i + 1$ node-disjoint paths. An *intruder* is a node that commits its resources to attacking the system, disrupting protocol runs and launching DoS attacks. It is under the control of an *attacker*, which denotes a human being trying to corrupt, i.e., control, nodes.

We assume that the system is synchronous as far as our algorithms are concerned, that is, we can give bounds on the message delay, processor speed and clock drift of the nodes. We feel that even in an ad-hoc network this is a viable assumption for nodes that are connected directly or through only a few intermediate relay nodes. Since we expect that nodes working together are also physically close together, we can usually use the synchronicity assumption. However, we will also consider what happens without it in Sections 4 and 7.

The transmission delay of a message depends on the route taken and we are given transmission delay bounds for the actual route taken. To avoid employing overly conservative worst-case bounds, we suggest measuring the delays regularly and using an appropriate statistic for computing the worst-case bounds. We also assume that if clock synchronization is available, then the difference between the clock values of any two nodes can be bounded by some known value π.

3 Hardware Support

Hardware support is pivotal in W$_2$F and is realized by means of a so-called "W$_2$F box" attached to every node. The box handles network access and contains all additional hardware features we require. It can change outgoing messages immediately before they are sent and filter incoming messages before they are passed to the software. The box is a trusted computing base and will at least feature some non-volatile storage which does survive a node crash. We also assume that the box does not disclose any secrets. Initially, the box knows the public/private key pair of the node and the public key of the certification authority.

The W$_2$F box is able to perform the following actions, which can be controlled individually by the software:

Timestamping: Messages are timestamped at the sender and the receiver by hardware. Timestamps are inserted on-the-fly, i.e., during the message transmission/reception operation, as close to the physical transmission as possible to exclude host latencies. This hardware mechanism has already been successfully used in the Network Time Interface (NTI) [7] developed in the course of our project SynUTC, which was dedicated to high-accuracy clock synchronization. If clock synchronization is not available on a node, then its hardware marks the timestamps as invalid. We will use the notation T_p^s resp. T_p^r to indicate the send resp. receive timestamp of node p.

Hashing: The W$_2$F box can on-the-fly compute the hash value of a message that is being transmitted or being received.

MACs (Message Authentication Codes): The W$_2$F box can on-the-fly compute a keyed one-way hash value as described in [8, p. 458]. The key is shared between sender and receiver. We will use $\sigma_p^h(m)$ to denote that node p has included a MAC for message m.

Signatures: We assume that our W$_2$F box can sign outgoing messages and that it is able to verify signatures of incoming messages. The send timestamp must be included in the signature, so precomputation and caching of signatures is not possible if hardware timestamping is turned on. We will use $\sigma_p(m)$ to denote that node p has signed the message m. A signature can also be requested by the software, in which case the signature is returned to it.

Encryption: The W$_2$F box can encrypt an outgoing message with the public key of the receiver if a fresh public key is available. We will use the notation $\epsilon_q(m)$ to indicate that a message has been encrypted with the public key of the receiver node q. The software can also request that a particular message is encrypted and delivered back to it. Of course, incoming encrypted messages can be decrypted as well.

Whenever we use signatures for authentication, we must protect ourselves against DoS attacks: Computing and verifying signatures are costly operations and can even take several milliseconds if performed in software [9]. Although we trust that hardware implementation will speed up both computation and verification time considerably, a node that verifies fake signatures is still wasting its resources.

In order to recognize and discard messages from intruders as soon as possible, the W$_2$F box contains the following hardware message filters, which can be turned on/off independently. Messages pass through the filters in the following order:

Receiver Filter: The W$_2$F hardware is capable of filtering out messages not addressed to the receiver. This is a well-established feature of network controllers.

Sender Filter: The hardware can also filter out messages from unwelcome senders. Such a feature is useful for filtering out messages from known intruders.

MAC Filter: If MAC filtering is active, then a message is only passed on to the software if the MAC could be verified. If there is no MAC key stored for the sender, then the message is discarded at once, otherwise it is discarded if the computed MAC does not match the one appended to the message by the sender.

Timestamp Filter: If the send timestamp and receive timestamp are both valid and the transmission delays are known, the W$_2$F box can filter out messages that were excessively delayed. Otherwise, messages are passed on unfiltered.

Signature Filter: If signature filtering is active, then a message is only passed on to the high-level protocol if the signature could be verified. If no public key is available for checking the signature, then the message is passed on unverified, but steps are taken to obtain a certificate and the recipient can request verification of the message at some later time.

Only messages which pass through the first four filter steps will actually require a time-consuming signature verification. The filters are ordered according to increasing resource consumptions at the receiver. An intruder can easily circumvent the address filters and will not have much problems with the timestamp filter, but the MAC and signature filters are very hard (virtually impossible) to bypass by intruders. The timestamp filter is only useful as long as timestamps are valid. If MAC filtering is active, then timestamp manipulations are recognized before the timestamp is checked. If signature filtering is active, then an intruder launching a DoS attack against the computationally expensive signature verification can circumvent the timestamp filter easily by setting the valid flag of the send timestamp to false. Nevertheless, the filter is useful for preventing accidental replays and should be used even if MAC filtering is not possible.

Of course, these filters could also be implemented in software. However, in this case we must contend with latencies generated by the operating system, which would extend the time during which a fake message must be kept in memory.

4 Authentication

We concur with the opinion stated in [4] and [3] that the attacker should be the first to commit its resources, which for our purposes consist of memory, computation time and signature verification time. The client puzzle approach of [4] is particularly attractive since it places practically all load on the client as the initiator of the communication, and allows the server to control the amount of effort required to establish the communication. The method is based on asking the client to solve the following equation by brute force:

$$h(C, N_S, N_C, X) = \underbrace{000\ldots000}_{k}Y$$

Here, h is a one-way hash function like SHA, C is the identity of the client, N_S is a nonce obtained from the challenge of the server, N_C is a nonce generated by the client for this puzzle, and X is the solution of the puzzle. The task is to find X so that the resulting hash value has k leading zeros. The remaining bits Y are not important. The difficulty of the puzzle is controlled by k, ranging from easy for small k up to impossible if k encompasses all bits of the hash.

Using this method would be a good choice for authentication except for several drawbacks: First, it at least quadruples the number of messages required, since every message from a client Alice to a server Bob is challenged. Secondly, the protocol has only been designed to protect servers. Alice must always authenticate messages from Bob and is thus vulnerable to DoS attacks while waiting for messages from Bob. Third, the method requires timestamps or some other freshness indicator and in an ad-hoc network we cannot guarantee that clock synchronization is available at all times. Finally, it does not allow for streamed communication. Alice would have to authenticate herself again with every new message. On the other hand, if we do not use something that is easily checked, we will end up checking faked signatures all the time. Therefore, we enhanced the protocol presented in [4] by including a nonce N_A in the first message sent by Alice and by adding a sequence number S_B to the already present timestamp in the puzzle message. It serves as a crude freshness indicator in case the clock synchronization fails. Although it will not prevent Alice from accepting an outdated puzzle, she will at least recognize a more recent puzzle and can then discard the old one. With the puzzle solution, Alice also includes a secret key encrypted with Bob's public key which is intended for MAC computation. If the solution is correct, Bob decrypts the MAC key and uses it for all further communication with Alice.

In all following protocols, $p \rightarrow q$: m means that node p sends a message m to node q. The term $[p, q, T_p^s, T_q^r]$ is added to the message by the hardware, where p is the sender, q is the receiver, and T_p^s resp. T_q^r is the send resp. receive timestamp of node p resp. q. Furthermore, N_p is a nonce generated by p, $\sigma_p(m)$ means that p's signature is appended to m, $\epsilon_p(m)$ signifies that m is encrypted with p's public key, K_{pq} is a key shared between p and q, and $\sigma_p^h(m)$ indicates that p has appended a MAC to m. The authentication protocol has the following form:

1. A $\rightarrow$ B: [B, A, T_A^s, T_B^r], N_A, `puzzle request`
2. B $\rightarrow$ A: [A, B, T_B^s, T_A^r], N_A, $\sigma_B(T_B, k, N_B, S_B)$
3. A $\rightarrow$ B: σ_A([B, A, $T_A'^s$, $T_B'^r$], N_B, N_A', X, $\epsilon_B(K_{AB})$)
4. B $\rightarrow$ A: σ_B^h([A, B, $T_B'^s$, $T_A'^r$], N_A', $Data_B$)

The first two messages of the authentication protocol are the puzzle request and reply. Bob sends both Alice's nonce and his (precomputed) puzzle in the second message. With the puzzle solution X, Alice sends Bob a shared secret key K_{AB} for further MAC computations. With the fourth message, Bob already includes a MAC and thereby acknowledges that he has received the MAC key. Bob may also include payload data $Data_B$ if the communication is two-way. In all further message exchanges, MACs are used for authentication.

After MAC authentication is established, only messages bearing acceptable or invalid timestamps will pass through the message filters, all other messages are discarded by the hardware during reception. Nevertheless, due to the uncertainties involved in clock synchronization and estimating message delays, there is still an opportunity for replays here, so the software responsible for the payload data should use sequence numbers if necessary.

As we have indicated, an intruder can launch DoS attacks against Alice since she must verify the signature of every message allegedly sent by Bob which passes through the timestamp filter. We have already explained that this filter can easily be circumvented if the intruder marks the send timestamp invalid. But as long as clock synchronization is available, we can make sure that an intruder cannot cause Alice to accept an outdated puzzle: If we assume that Bob creates puzzles with period P_{puzzle}, then we have two consecutive puzzles with times T_B and $T_B + P_{puzzle}$. Ignoring clock drifts to simplify the equations, Bob as the generator of the puzzle should accept solutions to a puzzle with timestamp T_B during the interval

$$[T_B + 2d_{min}, T_B + P_{puzzle} + 2d_{max} + \pi + T_k),$$

where d_{min} is the minimum message delay to/from Bob, d_{max} is the maximum delay, π is the clock synchronization precision, and T_k is the maximum computation time for a puzzle of difficulty level k. There is a time of overlap during which Bob must accept solutions to an old puzzle even though he has already generated a new one. But this is no DoS attack opportunity since Bob only stores data for correct puzzle solutions.

If Bob sends a fresh puzzle with time T_B to Alice, then Alice will receive and accept this puzzle at her local time T_A^r somewhere in the interval

$$T_A^r \in [T_B + d_{BA}^- - \pi, T_B + P_{puzzle} + d_{BA}^+ + \pi),$$

where d_{BA}^- resp. d_{BA}^+ is the minimum resp. maximum message delay from Bob to Alice. Although her acceptance interval for the next puzzle at time $T_B + P_{puzzle}$ will overlap with this one, creating a window during which both puzzles are valid, Alice should simply solve the first puzzle she receives, even if she obtains a more recent puzzle while working on the outdated one. Bob's acceptance window ensures that the outdated puzzle is nevertheless accepted.

If clock synchronization is not available, however, then Alice must accept every puzzle sent to her and will always have to solve the puzzle with the highest sequence number. Therefore, she must discard a puzzle she is working on if she receives a puzzle with a higher sequence number. Bob's window of acceptance will remain the same, but with π set to zero if Bob's timestamp is invalid.

Since we rely on timestamps and sequence numbers, we must consider what happens after a wrap-around. Fortunately, this is not a pressing problem as long as keys are changed more frequently than the wrap-around period: Since we are prepared for transient losses of clock synchronization anyway, the wrap-around of the timestamp itself is of no particular concern. There is, however, the problem that for every point in time in the new timeline, there are puzzles which were acceptable in previous timelines. The particular time at which a wrap-around occurs depends on the protocol used. NTP, e.g., has a 136-year cycle and the next wrap-around will happen in the year 2036 [10]. As far as sequence numbers are concerned, if we assume that they are stored in a 16 bit variable and if Bob generates a new puzzle every hour as suggested in [4], then we obtain an 8-year cycle before the sequence number wraps. However, Bob should store his puzzle sequence number in non-volatile memory to make sure he can recover it after a node crash.

Note that whereas we can safely assume that a node always has enough certification authority servers in its vicinity to rely on a synchronous subsystem, we cannot always be sure that Alice and Bob are within such a subsystem. We argue that it is the general case, but if this assumption is violated then our assumptions on network connectivity and transmission delay bounds may not hold. But due to the connection to the certification authority, both nodes at least get the fresh keys of the peer, so authentication is not at risk. There is, however, more opportunity for DoS attacks since the communication may be disrupted for an arbitrary amount of time. Therefore, the windows for DoS attacks on Alice will become much larger. What is more, Alice's solution to a puzzle may be delayed long enough to arrive only after Bob has already discarded the puzzle. Although Bob should choose his puzzle generation period P_{puzzle} large enough to minimize the chances of such a thing happening, nevertheless Alice may have to try repeatedly.

5 Key Publication

We will assume the existence of a trusted certification authority (CA) which stores and issues public key certificates for all nodes. We assume that the CA is distributed and fault-tolerant like e.g. COCA [11]. During key publication, nodes request public key certificates from the CA. In order to avoid constant polling of the CA whenever a message arrives, we chose a hybrid approach. If Bob needs to check a single message from Alice and does not have a fresh key, he requests Alice's key certificate from the CA. The request does not include signatures, we just assume that the CA can handle many requests and will start sending puzzles if it becomes overloaded. The protocol for a single query thus consists of just two messages:

1. B $\rightarrow$ CA: [CA, B, T_B^s, T_{CA}^r], A, N_B, `request certificate`
2. CA $\rightarrow$ B: [B, CA, T_{CA}^s, T_B^r], $Cert_A(K_A) = \sigma_{CA}(A, K_A, \nu_A)$

In this simple query protocol, no signature is required from Bob under normal circumstances since the CA has a lot of capacity anyway and can afford to answer all requests. The reply message itself is not signed to allow the CA to precompute the certificate. If, however, the CA gets overloaded or if it suspects that a DoS attack has been launched, then it will reply with a puzzle instead of the certificate. For this reason, Bob includes a nonce N_B with the request and stores it until he receives a message from the CA.

Instead of a simple request, Bob can send a subscription request, which makes the CA send Bob the new certificate whenever Alice's key changes. The subscription request must be signed. Bob can also use this protocol to cancel a subscription or to guaranteedly receive a fresh certificate (the first two messages are the same as in the authentication protocol of Section 4):

3. B $\rightarrow$ CA: σ_B([CA, B, T'^s_B, T'^r_{CA}], N_{CA}, N'_B, X, A, `once|subscribe|stop`)
4. CA $\rightarrow$ B: σ_{CA}([B, CA, T'^s_{CA}, T'^r_B], N'_B, $\mathrm{Cert}_A(\mathrm{K}_A)$)

4'. CA $\rightarrow$ B: σ_{CA}([B, CA, T'^s_{CA}, T'^r_B], N'_B, `subscription stopped|denied`)

In the third message, Bob sends the puzzle solution along with his request (single certificate request, new subscription or cancellation of an existing one). Since Bob will only send such requests sporadically, we can afford to use the puzzle approach every time and do not bother to negotiate a shared MAC key. In the final message (4), the CA sends Alice's current public key certificate to Bob if he has requested a single message or a new subscription. The message includes Bob's nonce and is therefore fresh. If Bob has subscribed, the certificate is from then on sent to him whenever Alice's key changes, but it does not include the nonce anymore since Bob can check the freshness of subsequent certificates via the version number ν_A. If Bob has cancelled a subscription, the CA sends an acknowledgement with message (4'). Here, the nonce is included to ensure Bob that the subscription has indeed been cancelled. Message (4') is also used in cases where for some reason the CA cannot process the subscription request (most likely due to overload).

The subscription feature is used to place the load of keeping Bob up-to-date with Alice's current key on the CA, which has most likely better hardware than Bob. In consequence, the nodes do not have to query the CA as often, so if key changes are not very frequent, then it relieves both the CA and the nodes from message traffic. In turn, it requires some memory for storing the subscription on the CA. If memory is of concern, the CA can force a node to resubscribe regularly by only storing subscriptions for a limited period of time.

The CA responds to Bob's request with Alice's current key certificate. Only the very first key certificate message is passed on to Bob's software, which determines whether the certificate should be processed and in that case orders the hardware to store the data from the certificate. As soon as a key is stored for a node, all subsequent certificates are processed within the W2F box, but only if the version number contained in the certificate is higher than the one currently stored. In that case, the box verifies the certificate and extracts the new key from it. The software protocol only has to decide whether the key for a particular node should be stored in the first place, and it also determines when the key of a node should be removed from the key table.

As long as changing a key is expensive, we do not see any attacks against the CA except that Bob can make the CA store his subscription and send messages to him whenever the key he has subscribed to changes. To minimize misuse, subscriptions should expire after some time.

The unsigned certificates sent by the CA in the unauthenticated version and as subscription updates allow an attack on Bob by sending old certificates until the fresh certificate from the CA comes through. But the duration and impact of this attack are limited. A more severe attack is to send faked certificates with a high version number. Bob has no defence against that and will check the signature of each such message. If Bob fears he is under attack, he should not react to certificate updates and should only use the authenticated single request method to obtain certificates.

6 Key Revocation

Frequent key changes help to prevent an attacker from guessing a fresh key with a brute-force attack. Therefore, all nodes should periodically change their keys and will ask the CA to sign and store their new certificates with the following protocol (the first two messages are the same as in the authentication protocol of Section 4).

3. A $\rightarrow$ CA: σ_A([CA, A, T'^s_A, T'^r_{CA}], N_{CA}, N'_A, X, K'_A, `key update`)
4. CA $\rightarrow$ A: σ_{CA}([A, CA, T'^s_{CA}, T'^r_A], N'_A, $\mathrm{Cert}_A(\mathrm{K}'_A)$)

Of course, if Alice's signature key has already been broken, then anyone can request a key update in Alice's name if they commit their resources to the puzzle. Since the new key pair or the certificate will be sent to Alice, she will at least know that her key has been broken and can raise an alarm. In the worst case, however, Alice's W2F box will have to be re-initialized.

Naturally, the CA should change its key as well. It can simply broadcast its new certificate to all nodes and then switch to the new key. Nodes who do not receive the certificate will notice the change with the next message of the CA and can request the new certificate then.

7 Assumption Violations

As we have already stated, W2F is based on an ad-hoc network. Therefore, our assumptions given in Section 2 may not always hold. Hence, we must consider what happens if they are violated. We have three assumptions:

(A1) The key of the certification authority has not been broken.

(A2) Nodes are connected directly or by $f_i + 1$ paths.

(A3) The system is synchronous.

Assumption (A1) is by far the most important of all since its violation affects the whole system. If an intruder breaks the key of the certification authority, then it can generate new certificates containing fake keys for any node at will, corrupting authentication at least until the next key update from the real CA comes through. At that point, corruption of the CA will at least be detected. Unfortunately, as long as we cannot identify and remove the intruders and all knowledge of the key, only manual change of the CA key and corresponding reconfiguration of all nodes will serve to re-establish authentication. Therefore, the CA should change its own key regularily.

If (A2) is broken, network partitions can occur. Nodes that are separated from the CA will not get new key updates and may work with old keys. To prevent a breach of security, nodes should only employ the authenticated certificate request protocol, which ensures that nodes either get a fresh certificate or none at all. Since separated nodes will not get the CA's public key as well, they must obtain it after rejoining the network. Reconfiguration after the rejoin must include high-level consensus mechanisms to establish the correct public key from the CA. A majority vote must be computed over the stored public CA keys of a sufficient number of nodes (at least $2f_i + 1$). But even a majority vote will not suffice if more than $n/2$ nodes were partitioned from the CA.

If (A3) is violated, then the timestamp filter is useless. As long as the violation is recognized, we can simply turn it off, but if the problem goes undetected, then valid messages are filtered out, creating an effect similar to violation of (A2) with all its consequences. In any case, turning the timestamp filter off will significantly increase the DoS attack opportunities during client puzzle exchanges.

8 Conclusions

We proposed a way to achieve authenticated message exchange in an ad-hoc network while keeping the possibilities for denial-of-service attacks at a minimum. We argued that even though the whole system may be asynchronous, we can nevertheless assume synchronous subsystems which contain enough servers of the certification authority to guarantee synchronous and fault-tolerant communication between nodes and the CA. We found that message filtering helps to keep DoS attacks at bay, although it does not completely prohibit DoS attacks in W2F. To minimize the response time of our filters, we implement them in hardware. In order to use our timestamp filter, nodes must know a lot about the system like message transmission delays or the global clock synchronization precision π.

As a next step, we must subject our algorithms to a formal analysis to prove their correctness. As experience has shown, even simple algorithms tend to contain subtle flaws which are only revealed through a rigorous formal verification. We will also evaluate the effectiveness of the denial-of-service defences employed by our algorithms using the cost-based framework of Meadows [6].

Acknowledgements

I am grateful to the anonymous reviewers for their valuable comments and suggestions.

References

[1] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[2] Lidong Zhou and Zygmunt J. Haas. Securing ad hoc networks. *IEEE Network*, 13(6):24–30, 1999.

[3] Kanta Matsuura and Hideki Imai. Modification of Internet Key Exchange resistant against denial-of-service. In *Pre-Proc. of Internet Workshop 2000 (IWS2000)*, pages 167–174, February 2000.

[4] Tuomas Aura, Pekka Nikander, and Jussipekka Leiwo. DOS-resistant authentication with client puzzles. In *Proceedings of the 8th International Workshop on Security Protocols*, LNCS 2133, pages 170–177, April 2000.

[5] Ari Juels and John Brainard. Client puzzles: A cryptographic countermeasure against connection depletion attacks. In *Proceedings of the 1999 Network and Distributed System Security Symposium (NDSS'99)*, 1999.

[6] Catherine Meadows. A cost-based framework for analysis of denial of service in networks. *Journal of Computer Security*, 9(1-2):143–164, 2001.

[7] Ulrich Schmid, Martin Horauer, and Nikolaus Kerö. How to distribute GPS-time over COTS-based LANs. In *Proceedings of the 31th IEEE Precise Time and Time Interval Systems and Application Meeting (PTTI'99)*, pages 545–560, Dana Point, California, December 1999.

[8] Bruce Schneier. *Applied Cryptography: Protocols, Algorithms, and Source Code in C.* John Wiley & Sons, second edition, 1996.

[9] Michael J. Wiener. Performance comparison of public-key cryptosystems. *CryptoBytes Technical Newsletter*, 4(1):1–5, Summer 1998. ftp://ftp.rsasecurity.com/pub/cryptobytes/crypto4n1.pdf.

[10] David L. Mills. NTP timescale and leap seconds. http://www.eecis.udel.edu/˜ntp/ntp_spool/html/leap.htm.

[11] Lidong Zhou, Fred B. Schneider, and Robbert van Renesse. COCA: A secure distributed on-line certification authority. Technical Report TR2000-1828, Computer Science Department, Cornell University, December 2000.