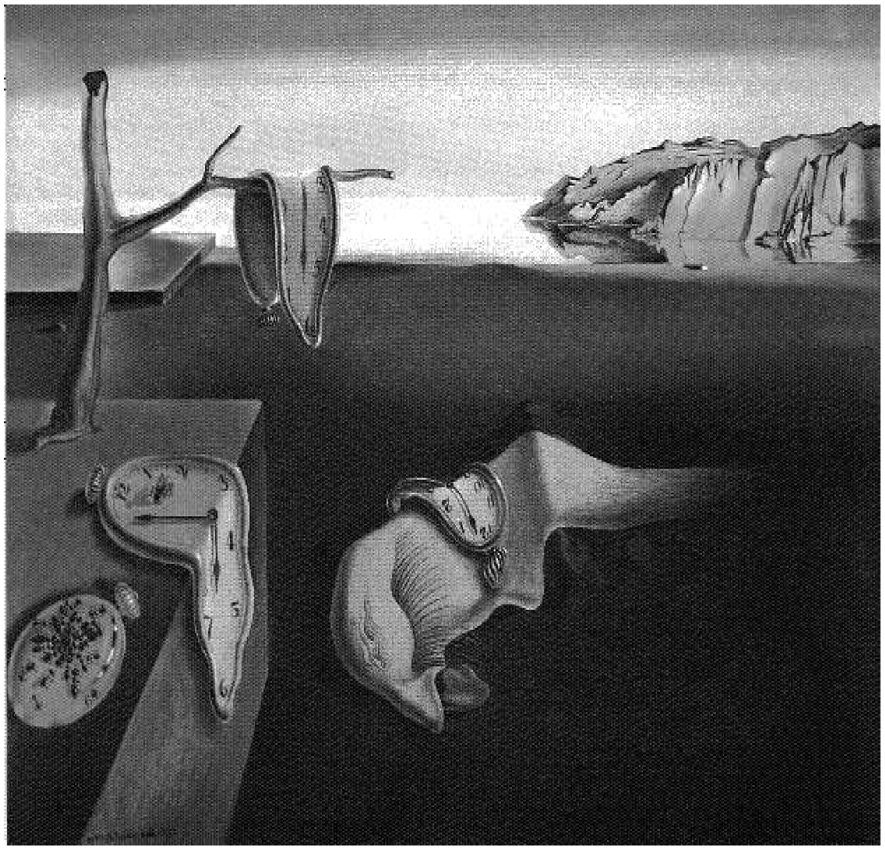
Institut für Automation
Abt. für Automatisierungssysteme

Technische Universität Wien

Projektbericht Nr. 183/1-122
July 2002

Authentication Under Denial-of-Service Attacks

Bettina Weiss


Salvador Dali, "Die Beständigkeit der Erinnerung"

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 get the public keys and how keys can be changed. We consider various methods of attack and their effect on our system. Although we can diminish the attack possibilities, we cannot completely protect our system from denial-of-service attacks due to its ad-hoc nature.

Keywords: *Fault-tolerant distributed systems, security, authentication, denial-of-service attacks, client puzzles, ad-hoc networks.*

1 Motivation

Over the last decades, security has become increasingly important for 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 [8, 5] profit from authentication.

The challenge of developing security algorithms is to make them attack proof. In the case of authentication algorithms, this means that an intruder should not be able to cause a node A to believe it is talking to B when in reality it is talking to some node C $\neq$ B. But what is equally important is to provide for 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 fake requests. DoS attacks may also suppress valid messages and thereby cause nodes to wait unduely long. Since W$_2$F is concerned with the quality of its services and with 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. Many papers specifically address solutions to the TCP SYN flooding attack [2], a frequently exploited vulnerability in the TCP/IP protocol which allows attackers to establish half-open connections

*This research is part of our W2F-project, which targets a wireline/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.

with a server, causing the server to create in its memory data structures for pending connection requests which are never completed. Since the server's memory is finite and the attacker may use a spoofed IP address to conceal its true identity, the server's memory may fill with these data structures and legitimate requests will be denied. After attacks in 1999 and again in 2000 on several large Internet sites including Yahoo and Amazon [4], efforts have increased even more to prevent such attacks in the future. The suggested solutions mainly aim at preventing address spoofing, e.g. through authentication, and tracing attack packets back to the sender [15] or at detecting and filtering out attacks [17].

Zhou and Haas [20] 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. Zhang and Lee [18] deal with intrusion detection techniques in wireless ad-hoc networks. In their proposed architecture, every node participates in the detection effort to detect abnormal activities. They assume the existence of a secure communication channel among nodes.

Matsura and Imai [10] 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 [1], where the authors use client puzzles [7] to make the client commit its resources first. The protocol does not require the server to store any client-related state until the client has committed itself. 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.

In this paper, we primarily investigate the benefits of integrating message filtering at the hardware level. Although this mechanism was found to be helpful for diminishing the effects of DoS attacks, it nevertheless cannot completely prevent them. We use signatures and message authentication codes to identify senders, thus keeping intruders from generating valid messages under false names. To foil attacks which replay old messages, we use timestamps whenever possible and sequence numbers as a fallback mechanism in cases where the timestamps are not available. Suppression of valid messages, finally, is countered by connection redundancy.

In the remaining paper, we will first present our system model in Section 2 and state our requirements to the hardware in Section 3. Section 4 will illustrate how the hardware mechanisms are used to diminish the feasibility of DoS attacks. In Sections 5, 6, and 7, we will propose algorithms for key publication, authenticated message exchange, and key revocation, and will informally analyse their resilience to DoS attacks. In Section 8, we consider what happens if our assumptions are violated. Section 9 concludes the paper.

2 System Model

We assume a set of n nodes interconnected by some ad-hoc network which ensures that any two nodes are either connected directly or by $f+1$ node- and link-disjoint paths for some $f \geq 0$. Nodes are either correct or intruders, with at most $0 \leq f_i \leq f$ intruders in the system. A path is correct if all relay nodes are correct. In the text below we will also use the term *attacker*, which denotes a human being trying to corrupt, i.e., control, nodes. A node that has been corrupted is regarded as an *intruder*. An intruder will commit its resources to attacking the system, disrupting protocol runs and launching

DoS attacks. A corrupted node should not be confused with a *compromised node*. The former is under the control of an attacker, the latter is a correct node whose secrets have been found out by the attacker and can be used by any intruder. We need to distinguish between these two types of nodes because an intruder node has a malicious intent, whereas a compromised node has no such inclinations and will even do its best to help the system to recover.

We assume that our system is basically asynchronous, but that nodes which are connected directly or via only a few relay nodes form a synchronous subsystem with known bounds on processor speed and clock drift of correct components. The message transmission delays in this subsystem depend on the route of the message and we postulate the existence of a known upper bound. In general, however, transmission delays are significantly lower than this bound and are measured online. While a message travels along a specific path consisting only of correct nodes, its transmission delays will be fairly constant. 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 value π.

We will assume that paths and thus transmission delays in a synchronous subsystem do not change too frequently and will regard them as constant for one of our protocol runs. If the paths are constant, this also means that messages from a single correct sender travelling along a correct path, i.e., a path consisting only of correct intermediate nodes, arrive at the receiver in the order they were sent. We demand that $f_i < f$ within a synchronous subsystem.

We assume that the system starts with only correct nodes and that there is no easy way for mass compromisation or corruption. Hence, an attacker has to spend some time on corrupting one node. If we can ensure that compromising one node does not make corrupting other nodes trivial then we can estimate the time it takes to corrupt or compromise f_i nodes. If we use intruder detection mechanisms as well and try to change keys faster than the attacker can break them, then we might be able to fight off the point where there are too many intruders and compromised nodes for the system to handle indefinitely.

The assumption that compromising one node does not make further compromisations trivial is rather grave: If by compromising or corrupting a node p the attacker finally gains control of our certification authority (CA), then all correct nodes will be compromised at the latest after the next key change. So the assumption effectively implies that the key of the CA must not be broken. If p is not part of the CA, then its corruption is less severe. Of course, the more nodes the attacker has under its control, the more processing power it has to find out the keys of correct nodes, but there is still time and work involved in finding out other keys.

To recapitulate, our assumptions on the (synchronous) system are:

- Any two nodes are connected via $f + 1 \geq 1$ disjoint connections.
- There are $0 \leq f_i \leq f$ intruders and compromised nodes in the system.
- Connections remain constant during one protocol run.
- The transmission delay bounds for a particular connection are known.
- The key of the certification authority is never compromised.
- If clock synchronization is available, then for any two correct nodes p and q with valid timestamps the inequality $|C_p(t) - C_q(t)| < \pi$ holds for some globally known π.

3 Hardware Support

The W2F security system is parted into a hardware and a software layer. The hardware layer is located between the network controller and the physical medium and can change outgoing messages immediately before they are sent. The software layer is directly above the message event handler and can filter any incoming message.

Hardware support is very important in our W2F project and it is realized with a so-called "W2F box" attached to every node. The box handles network access and contains all additional hardware features we require. It is a trusted computing base and will most likely use a secure coprocessor [14]. At the least, we require some non-volatile storage space which does survive a node crash and assume that the box does not disclose any secrets (namely, the secret keys of the node). If the secrets of the box are disclosed, the owner node is regarded as compromised. Corrupting the hardware over the network must be prevented, hence an attacker actually has to gain physical possession of the box to corrupt it. Since we envision W2F to be a fieldbus which is limited to the confines of a factory or some other restricted area, this already reduces the number of attackers able to gain access to a box. Nevertheless, the final W2F box will most likely feature a secure coprocessor with tamper-proof or at least tamper-aware memory. Initially, the box knows the public/private key pair of the node, the public key of the certification authority, and two public values n and g necessary for key revocation.

In W2F, we have clock synchronization available. However, since we are in an ad-hoc network where the degree of connectivity is not guaranteed, clock synchronization depends on authentication to minimize the capacity of byzantine faulty nodes. In consequence, we will assume that a timestamp T taken at real-time t contains the local clock time $C(t)$ and a flag stating whether the time is valid or not. If the timestamps T_p and T_q of two nodes p, q are both valid and the nodes are both correct, then $|C_p(t) - C_q(t)| < \pi$ where π is the global precision of the clock synchronization algorithm. If the flags are not valid, then the timestamps are arbitrary and bear no relationship. Every node is in this state immediately after joining the W2F network for the first time, and a node may fall back into the unsynchronized state if the synchronization of the clock cannot be guaranteed (most likely due to sparse network connectivity). Apart from the global precision π which is valid for all nodes, clock synchronization can also guarantee a precision $\pi_S \leq \pi$ within a synchronous subsystem S if the connectivity of the subsystem is sufficient. In this paper, however, we will only use the global precision π.

Note that in W2F there is a symbiotic relationship between authentication and clock synchronization: Our authentication algorithms can live without clock synchronization, but if it is available then denial-of-service attacks are severly restricted and can be filtered out by hardware most of the time. Clock synchronization, on the other hand, requires $2f_i + 1$ disjoint (relayed) connections between nodes without authentication and just $f_i + 1$ such connections if authentication is available. So the proper startup sequence will be to first establish authentication without synchronized clocks, thereby perhaps even just allowing clock synchronization to kick in, and then use the synchronized clocks to keep attackers at bay. Obviously, the system is most vulnerable to DoS attacks during the phases where clock synchronization is not available.

The W2F box is able to compute a hash on-the-fly. We propose to use RIPE-MD or an equally secure algorithm that is fast enough. According to Schneier [13, Table 18.2, p. 456], RIPE-MD achieves an encryption speed of 1.42 Mbit/s as a software implementation on a 33MHz 486SX. We assume a W2F data transmission speed of 1 Mbit/s, so any hash algorithm which achieves at least 1 Mbit/s in the hardware implementation can be used for hashing messages on-the-fly. Since the hash function is also used to generate message authentication codes (MACs) [16], the hash function should be slightly faster than 1 Mbit/s. If we use the mechanism described in [13, p. 459], where at least 64 bits of the key are appended to every message block, and if we assume a message block size of 512 bit as used by MD5 and SHA, then the hashing function must have a speed of at least (64+512)/512 = 1.125 Mbit/s.

The W2F box can sign messages using a public-key signature scheme. To achieve this, we assume that there is a (hardware-)programmable table of public keys with entries of the form $\langle p, K_p, \nu_p \rangle$ from where the hardware can take the public key K_p with version number ν_p of node p during signature verification. We also assume that the hardware is capable of recognizing new key certificates signed by the certification authority and can update the table accordingly. The hardware takes over a new certificate $Cert_p(K'_p)$ consisting of the key K'_p and the version number nu'_p for a node p if it has currently stored the data from a certificate $Cert_p(K_p)$ and if the new version number fulfills $\nu'_p > \nu_p$.

We do not commit ourselves to any specific public key encryption method, we just require that the keys should be reasonably hard to find out. We recommend using a scheme where both signature computation and verification are fast (at least when implemented in hardware), even if it means that the scheme is less secure and we have to change keys more often. We place, however, more weight on the signature computation speed than on the verification speed.

To sumarize, the W2F box has the following features. The first list contains modifications to messages sent by a node, whereas the second list consists of diverse message filtering techniques at the receiver. The filters can be selected independently of each other on a per-node basis, but if they are used, then they assume that their respective hardware mechanisms are turned on.

Timestamping: Messages are timestamped automatically by hardware at the sender and the receiver. Care is taken to insert the timestamps as close to the physical transmission as possible to exclude host latencies. The timestamps are inserted into reserved fields of the message on-the-fly, i.e., during the message transmission/reception operation on the physical layer. This hardware mechanism has already been successfully used in our project SynUTC, which was dedicated to high-resolution clock synchronization [6].

Hashing: The W2F box can on-the-fly compute the hash value of a message that is being transmitted or being received.

MACs (Message Authentication Codes): The W2F box can on-the-fly compute a keyed one-way hash value as described in [13, p. 458]. If the method is sufficiently secure or if the hash computation is slow, the hash is computed to be $h(K, p, m, K)$ where K is the key, p pads K to a full message block, and m is the message itself. If the hash function is fast enough, then we can use the method where at least 64 bits of the key are concatenated with each message block. In both cases, the key is shared between sender and receiver. We will use $\sigma^h_p(m)$ to denote that node p has included a MAC for message m.

Signatures: We assume that our W2F hardware can sign messages “on-the-fly” during the send operation and that it is able to verify signatures of incoming messages “on-the-fly” during the receive operation if a fresh, i.e., a currently used, public key of the sender is available to the hardware. 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 W2F box can encrypt an out-going message with the public key of the receiver if a fresh public key is available. We will use the notation $\sigma_q^{-1}(m)$ 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.

Note that the signature mechanism is not as on-the-fly as we made it sound. Computing and verifying signatures are costly operations and can even take seconds if performed in software [13, p. 488, Table 20.2]. However, we trust that hardware implementation will speed up the computation and verification time considerably, so we expect them to be at most in the 1-10 ms range, probably even lower for a fast and less secure algorithm than the DSA algorithm measured in [13]. Secondly, we can measure the signature computation time by inserting a second timestamp at the end of the message immediately before signature computation is completed. This allows the receiver to exactly measure the signature computation delay, which is then simply added to the transmission delays the receiver attributes to the message. So we allow the hardware to delay the message for the time necessary to compute the signature —this time is not contained in the send timestamp—, and the time which was spent for signature computation is added at the receiver.

Verification also takes time, but since it is spent at the receiver, it does not influence the filtering mechanism. It is, however, a weak point for a DoS attack. Nevertheless, our emphasis is on the speed of the signature, it should be as fast as possible.

The W2F 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 W2F hardware is capable of filtering out messages not addressed to the receiver. This is a well-established feature of network controllers. The filter uses a simple comparison of the receiver's name with the receiver address field of the message and discards the message if there is no match.

Sender Filter: The hardware can also filter out messages from unwelcome senders. Such a feature is useful for filtering out messages from known intruders. The list of unwelcome senders must be accessible to the hardware. Every entry in the list is compared to the sender address field and the message is discarded as soon as a match is found.

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: In a synchronous subsystem where message delays are known, the W2F hardware can filter out messages which were excessively delayed. If we assume that we know the message transmission delay interval $d_{pq} = [d_{pq}^-, d_{pq}^+]$ between two nodes p and q and if we know the upper and lower bounds on the clock drift $\rho_q^\pm$ of receiver q, then if q's timestamps are valid, a message from p containing a valid send timestamp T_p will only be passed on to the higher layers if $C_q(t_r) \in [C_p(t_s) - \pi + d_{pq}^-\rho_q^-, C_p(t_s) + \pi + d_{pq}^+\rho_q^+]$, where $C_q(t_r)$ is the receive timestamp of node q. Of course, this feature can only be used if the transmission delay bounds are known. Since there are $f + 1$ possible paths between any two nodes, it follows that the receiver must know the path a message took to know what transmission delay bounds to use. If all paths have similar delays, then it may be possible to use a common transmission delay interval that includes all possible paths. As we stated previously, transmission delays are measured online and are required both by clock synchronization and by our locationing service, so we can postulate that this information exists. If we do not have the information, or if either the send or the receive timestamp is invalid, then messages are passed through 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 fresh public key is available for checking the signature, then the message is passed on unverified, but steps are taken to obtain a fresh key. The recipient can then 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.

4 Denial of Service Prevention With Client Puzzles

Basically, we concur with the opinion stated in [1] and [10] that the attacker should commit its resources, which for our purposes consist of memory, computation time and signature verification time, first. The client puzzle approach of [1] is particularly attractive since it places practically all load on the client. The method is based on asking the client to solve the following equation by brute force:

$$h(C, N_S, N_C, X) = \underbrace{000\ldots000}_{k}Y \tag{1}$$

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.

To summarize, the method works as follows (we assume that Alice contacts Bob to emphasize that the method can be used for any two nodes, not just a client and a server): Periodically, Bob precomputes and signs a message $\sigma_B(T_B, k, N_B)$ consisting of a timestamp, a nonce, and the difficulty parameter k. When Alice contacts Bob, the following exchange takes place.

1. A $\rightarrow$ B: Request
2. B $\rightarrow$ A: σ_B(T$_B$, k, N$_B$)
3. A $\rightarrow$ B: σ_A(B, A, N$_B$, N$_A$, X)
4. B $\rightarrow$ A: σ_B(A, B, N$_A$)

After receiving the third message, Bob makes certain that the triple A, N_B, N_A has not been used before and verifies the solution X. After that, he may commit his own resources to the connection and will store A, N_B, N_A as long as N_B is fresh. He also verifies the signature σ_A to authenticate Alice. If Alice needs to authenticate Bob as well, then Bob should send the last message.

Using this method would be a good choice for authentication except for three drawbacks: It at least triples the number of messages required, since every message from Alice to Bob is challenged, and only after the reply does Bob react to the contents of the message. If Bob broadcasts his challenge from time to time, then there are only two messages (three if Alice needs to authenticate Bob) involved, but the network load in our system would be even higher due to the broadcast. 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 2 and 4. Although we can reduce the vulnerability until message 2 by requiring that Alice includes a nonce in the first request which is sent back by Bob, an attacker knowing the nonces can still start DoS attacks. Third, the method requires timestamps or some other freshness indicator. We cannot guarantee that clock synchronization is available at all times. Therefore, we do not wish to use this kind of authentication for the whole time. On the other hand, if we do not use something that is easily checked, we will end up checking fake signatures the whole time. Hence, we will use the following combination of techniques for authentication:

If Alice has never talked to Bob before or if there has been a period of quietness which was long enough for Bob to remove all communication data stored for Alice, then we use the client puzzle approach. But in the third message, Alice also includes a secret key encrypted with Bob's public key which is intended for MAC computation. After receiving the message and checking the correctness of the solution, Bob finally commits his own resources by storing Alice's public key and verifying the signature. Bob also decrypts and stores Alice's MAC key. The fourth message does already include a MAC and serves as an acknowledgement that Bob has received the key.

From then on, Bob turns on MAC filtering for Alice. The hardware can even recognize this state automatically by the fact that a shared MAC key is stored for Alice. All further communication with Alice is now authenticated with MACs. Since shared keys are normally considered to be relatively insecure, Alice should change the key from time to time. If Bob does not get a new key within a certain time, he will fall back to the initial phase and send Alice a client puzzle again.

The general authentication protocol has the following format (items in braces are inserted automatically by the hardware):

1. A $\rightarrow$ B: [B, A, T^s_A, T^r_B], N_A, `puzzle request`
2. B $\rightarrow$ A: [A, B, T^s_B, T^r_A], N_A, $\sigma_B(T_B, k, N_B, S_B)$
3. A $\rightarrow$ B: σ_A([B, A, T'^s_A, T'^r_B], N_B, N'_A, X, Data_A)
4. B $\rightarrow$ A: σ_B([A, B, T^s_B, T^r_A], N'_A, Data_B)

In addition to the nonce sent by Alice in the first message, we also include a sequence number S_B with 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. The $Data$ items either contain node data in case of a single message exchange, or $Data_A$ contains the encrypted MAC key used for MAC filtering in subsequent communication, see Section 6.

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

$$T_A^r \in [T_B - \pi + d_{BA}^-(1 - \rho_A^-), T_B + P_{puzzle} + \pi + d_{BA}^+(1 + \rho_a^+)). \tag{2}$$

However, her acceptance interval for the next puzzle at time $T_B + P_{puzzle}$ will overlap with this one, creating a window

$$[T_B + P_{puzzle} - \pi + d_{BA}^-(1 - \rho_A^-), T_B + P_{puzzle} + \pi + d_{BA}^+(1 + \rho_a^+)] \tag{3}$$

during which both puzzles are valid. Here, Alice should simply solve the first puzzle she receives, even if she obtains a more recent puzzle while working on the outdated one.

As a consequence, Bob as the generator of the puzzle must accept solutions to a puzzle with time-stamp T_B during the interval

$$[T_B + 2d_{min}(1 - \rho_B^-), T_B + P_{puzzle} + \pi + (d_{max}(1 + \rho_A^+)(1 - \rho_A^-) + T_k)(1 + \rho_B^+)), \tag{4}$$

where $d_{min} = min_A\{d_{BA}^-\}$, $d_{max} = max_A\{d_{BA}^+\}$, and T_k is the maximum computation time for a puzzle of difficulty level k. There is again 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.

Since Bob allows this overlap and includes Alice's uncertainty π into it, Alice is safe from DoS attacks via old puzzles since the solution to every puzzle she accepts as valid will also be accepted by Bob as long as Alice does not take more time than T_k to solve the puzzle.

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. In the case of timestamps, it is clearly the task of clock synchronization to anticipate the wrap-around and mark timestamps as invalid until the precision condition is fulfilled again. 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. If Alice's timestamp and the timestamp of the old puzzle are both valid, Alice will accept the old puzzle and Bob will discard the solution. For this attack to work, the intruder must have the old puzzles (in fact, as large a sequence of them as possible to make more than a brief impact), timestamps must be valid in the old puzzles and now, the old timestamps must match Alice's acceptance windows, the signature must still be valid, and the intruder must know the current nonce Alice has sent to Bob. The attack lasts as long as the intruder has old puzzles with suitable timestamps and signatures. Although possible, we deem the chances for this attack to occur to be rather low. The particular time at which a wrap-around happens depends on the protocol used, NTP, e.g., has a 136-year cycle and the next wrap-around will happen in the year 2036 [12].

Sequence numbers are a fallback mechanism in case timestamps are invalid. Here, a wrap-around has severe consequences in that an intruder can replay any previous puzzle containing a higher sequence number than the current one and thus can cause Alice to reject the correct puzzle. However, this only works if the signature of the old puzzle is still valid, i.e., if Bob has not changed his key since then. For the attack to work, the intruder only has to store one old puzzle, preferably with an invalid timestamp, that has a high sequence number, and can then prevent Alice from solving the correct puzzle until clock synchronization is available. Fortunately, the problem is not too pressing: If we assume that sequence numbers are stored in a 16 bit variable and if Bob generates a new puzzle every hour as suggested in [1], then we obtain an 8-year cycle before the sequence number wraps. Bob should change his key more frequently anyway. However, Bob should store his puzzle sequence number in non-volatile memory to make sure he can recover it after a node crash.

5 Key Publication

If we use a public key system, then we must assume that a trusted certification authority (CA) knows the public keys and can verify the binding of public keys to nodes. We assume a distributed CA like COCA [21, 19]. Note that our system assumptions concerning links are stronger than those of COCA in that we assume at least one correct path, whereas COCA only requires links to be fair, i.e., if infinitely many messages are transmitted, then infinitely many messages are correctly delivered. Apart from that, they assume a fully asynchronous system, whereas we require a synchronous subsystem to implement our denial-of-service defences. The assumptions complement one another nicely, since the CA itself is distributed over the whole network and hence cannot depend on synchronicity, whereas a node communicating with the CA does so by talking to nearby servers that are part of a local subsystem in which the synchronicity assumption is viable. In both our system and COCA, if clients are partitioned for too long, then either keys must be considered as out-dated and system operation is disrupted, or clients must risk using compromised keys.

In key publication, our concern is how to get fresh public keys to the nodes who need them. In order to avoid constant polling of the certification authority 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 overloaded. Instead of a single request, Bob can also send a subscription request, which makes the CA send Bob the new certificate whenever Alice's key changes. The subscription request must be signed.

The protocol for a single query consists of just two messages:

1. B $\rightarrow$ CA: [CA, B, T^s_B, T^r_{CA}], A, N_B, `request certificate`
2. CA $\rightarrow$ B: [B, CA, T^s_{CA}, T^r_B], $\mathrm{Cert}_A(K_A) = \sigma_{CA}(A, K_A, \nu_A)$

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.

If Bob wants to subscribe to or cancel certificate updates, or if he wants to be sure that the certificate he receives is fresh, then he must authenticate himself and therefore must request a puzzle first, resulting in the following protocol:

1. B $\rightarrow$ CA: [CA, B, T^s_B, T^r_{CA}], N_B, `puzzle request`
2. CA $\rightarrow$ B: [B, CA, T^s_{CA}, T^r_B], N_B, $\sigma_{CA}(T_{CA}, k, N_{CA}, S_{CA})$
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, $\text{Cert}_A(K_A)$)

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

In the first message, Bob sends his request to the CA and the CA replies with the puzzle. Bob sends the solution in the third message, 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. 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. Upon reception, Bob's W2F box checks whether the version number contained in the certificate is higher than the one currently stored, and in that case verifies the certificate and extracts the new key from it. Processing is done automatically by the hardware to relieve higher layers from a task that ultimately results in the hardware processing the certificate anyway. 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.

Since the key of the CA is extremely important and must not be compromised at all costs, the CA itself should change its key from time to time. It will broadcast its new public key (using the old key to sign the message) to all nodes in the network. As long as every node is connected over a correct path to the CA, every node will get the new key and will update its key table. After the broadcast, the CA can use the new key.

Apart from DoS attacks on Bob during the puzzle request phase, which we have described in Section 4, the second DoS attack opportunity on Bob is while he waits for the final message from the CA which Bob has to authenticate again. This corresponds to the attack opportunity described for the general puzzle protocol. We do not see any other DoS opportunities here, in particular, we do not see any attacks against the CA except that Bob can make the CA store his subscription (two names) 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. This is only a problem if Bob does not already have a key stored for Alice, otherwise old certificates are filtered out without any signature checks. If no key is stored, then the certificate is passed on to the software layer, where it is filtered out if the corresponding node is of no interest and else stored if it is.

A severe attack on the authentication itself can be launched if the certificate version number wraps before the CA key changes, because now the intruder can send any old certificate with a higher number. But as we have argued for the sequence numbers described in Section 4, the cycle for a wrap-around is much longer than the key change rate anyway.

6 Authentication

Let us assume that Alice and Bob want to talk and that they already know the fresh public key of the peer. Then we will use the algorithm sketched in Section 4 for authentication:

1. A $\rightarrow$ B: [B, A, T^s_A, T^r_B], N_A, `puzzle request`
2. B $\rightarrow$ A: [A, B, T^s_B, T^r_A], N_A, $\sigma_B(T_B, k, N_B, S_B)$
3. A $\rightarrow$ B: σ_A([B, A, T'^s_A, T'^r_B], N_B, N'_A, X, $\sigma_B^{-1}(K_{AB})$)
4. B $\rightarrow$ A: σ^h_B([A, B, T'^s_B, T'^r_A], N'_A, Data_B)

In the first two messages, Alice requests a puzzle and Bob sends it to her. In the third message, Alice sends Bob a shared secret key for further MAC computations. Bob stores the key, thus telling his hardware that all further messages to Alice should include a MAC. After sending the third message, Alice also stores this key, enabling her hardware to verify all further MAC messages sent by Bob. With the fourth message, Bob acknowledges that he has received the MAC key. Bob may also include payload data if the communication is two-way. In all further messages exchanges, MACs are used for authentication.

For one-way communication, Alice will have to store the following parameters for Bob:

- Bob's name
- Bob's public key (only if data should be encrypted)
- shared MAC key (if MACs are used)

After Alice has been authenticated and the communication request has been granted, Bob as the receiver will also have to store some parameters:

- Alice's name
- filter mode
- Alice's public key (if signatures are used)
- shared MAC key (if MACs are used)
- delay parameters per path from Alice to Bob (if timestamp checking is used)

In a two-way communication, Alice stores the same items as Bob.

All DoS attacks on the puzzle request have already been described in Section 4. We see that in a one-way communication, Bob as the receiver has to store more data than Alice as the sender. Alice can even renounce timestamp filtering and just use a software filter on the nonce to discard replayed messages from previous protocol runs while waiting for Bob's last reply. She also does not have to store a filter mode, since the hardware automatically uses MAC filtering if the message type indicates a MAC and if an appropriate key is stored. Nevertheless, an intruder does not gain anything from establishing an unnecessary connection with Bob since the intruder will have to solve the puzzle first and will be challenged with a puzzle again if Bob starts to become overloaded.

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, there is an opportunity for replays here, so the software responsible for the payload data should use sequence numbers if necessary.

Note that whereas we can safely assume that a node always has enough CA 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 we must also consider what happens if Alice and Bob are so far apart that we must assume an asynchronous system. In this case, our assumptions on network connectivity and transmission delay bounds may not hold. The security of the authentication protocol is nevertheless guaranteed: Since both Alice and Bob have a working connection to the CA, they can get the fresh public key of the peer to verify the signatures. And since only Bob can decrypt the MAC key Alice has sent to him, all further communication using MACs is also authenticated. There is, however, more opportunity for DoS attacks since 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 communication may be interrupted for an arbitrary long time.

7 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. Here, we are faced with the dilemma of who to entrust with generating the new keys. If we place the responsibility on the CA, then we can be sure that it will select good, i.e., cryptographically secure, keys. **Note:** *The problem was mentioned in some paper. But which one???* However, if just one server of the CA is corrupted, it may find out the secret keys of all nodes. On the other hand, if we let the nodes decide their own keys, then not every node may have enough computing power to select a good key. Still, it is a trade-off between some small nodes using insecure keys and having to ensure that no server node of the CA is ever corrupted to secure the keys of all nodes. We strongly suggest to let nodes choose the keys for themselves, but nevertheless we provide algorithms for both variants.

In both cases, if Alice wants to request a key update, then she must solve a puzzle. Therefore, the first two messages are the same for both protocols:

1. A $\rightarrow$ CA: [CA, A, T^s_A, T^r_{CA}], N_A, `puzzle request`
2. CA $\rightarrow$ A: [A, CA, T^s_{CA}, T^r_A], N_A, $\sigma_{CA}(T_{CA}, k, N_{CA}, S_{CA})$

If new keys are issued by the CA on the basis that the CA has enough computing power to select good keys, then if Alice wants to change her key, she must request a key update from the CA. Even if Alice is still uncompromised, the CA cannot use her current public key to encrypt the new key: An attacker who finally cracks one old key and who possesses all subsequent key change messages sent by the CA can then read the current key as well. Therefore, we must resort to a different key shared between the CA and Alice. We propose to use the Diffie-Hellman [3], [13, p. 513ff.] key-exchange protocol for computing the shared secret key between Alice and the CA. The protocol assumes that two parties select a large prime n and some value g that is primitive mod n. These two values are public knowledge. To establish a common shared key between Alice and Bob, Alice chooses a large random number x and sends the value $X = g^x$ mod n to Bob. Bob, in turn, selects some large random number y and sends Alice the value $Y = g^y$ mod n. Since $k = Y^x = X^y = g^{xy}$ mod n, Alice and Bob now share a key k that cannot be computed by anyone else.

Note: *Instead of the exponentiation, we could also use elliptic curves since the key size is smaller and thus the algorithm is faster, see [9].*

To integrate this protocol into our system, we assume that Alice selects some large random number x prior to sending the key update request and sends g^x mod n to the CA together with the puzzle solution. The CA selects some value y, computes the shared key g^{xy} mod n, and sends its public value g^y mod n together with Alice's encrypted new key. Upon reception, Alice uses g^y to compute the shared key and decrypts her new key pair. The new key pair is loaded into the W_2F box.

3. A $\rightarrow$ CA: σ_A([CA, A, T'^s_A, T'$^r_{CA}$], N_{CA}, N'_A, X, g^x mod n, `key update`)
4. CA $\rightarrow$ A: σ_{CA}([A, CA, T^s_{CA}, T^r_A], N'_A, g^y mod n, $\{ K'_A, K^{-1}_A{}' \}_{K_{A,CA}}$)

With the solution of the puzzle, Alice also sends her key update request. For the CA to authenticate Alice with the third message, we must assume that Alice's key is still uncorrupted. In the last message, the CA replies with a new public and private key pair for Alice.

As soon as Alice sends her public part of the shared secret key, an attacker can work on finding her secret value x. If the attacker finds that value, it can compute the shared key and read Alice's secret key. Therefore, we must require that Alice requests a new update earlier than that. Note that decrypting an old key update reply does not help the attacker to find recent key values, since Alice and the CA use a new random key with every update request.

If Alice is allowed to change the key by herself, then she just wants the CA to sign and store her new public key certificate and no shared key is required.

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, $Cert_A(K'_A)$)

In both variants, if Alice's signature key has already been compromised, 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 compromised and can raise an alarm. In the worst case, however, Alice's W2F box will have to be re-initialized.

8 Assumption Violations

As we have already stated, W2F is based on an ad-hoc network. Therefore, our assumptions given in Section 3 may not always hold. Hence, we must consider what happens if they are violated.

(A1) The key of the certification authority is never compromised.

(A2) Any two nodes are connected via $f + 1 \geq 1$ disjoint connections.

(A3) There are $0 \leq f_i \leq f$ intruders in the system.

(A4) Connections remain constant during one protocol run.

(A5) The transmission delay bounds for a particular connection are known.

(A6) For any two nodes p and q with valid timestamps the inequality $|C_p(t) - C_q(t)| < \pi$ holds for some globally known π.

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, we fear that only manual change of the CA key and corresponding reconfiguration of all nodes will serve to re-establish authentication.

If (A2) or (A3) are broken, then we have a big problem with the key revocation of the CA's key: If a node p is partitioned —either through violation of (A2) or because the intruder nodes suppress all messages— from the rest of the system long enough for an intruder to compromise the last certification authority key K_S known to p, then the intruder can from then on generate an update message containing a fake new CA key. After p has regained access to the correct CA, it will not recognize the current CA key anymore. However, it also cannot simply take over a key which was sent to it from the real CA, but must use high-level consensus mechanisms to establish which public key from the CA is the correct one. 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.

Violation of (A2) or (A3) also poses a lot of problems for the algorithms since messages may not arrive at the destination. This is no problem during the puzzle request phase except that it disrupts communication. In the key publication protocol, however, if Bob has subscribed, a sudden partition will go unnoticed until Bob's subscription runs out. So the authentication protocol may not be secure anymore if (A2) or (A3) are violated between the CA and the nodes. Note that there is no problem with authentication between a node and the CA since the CA always knows the fresh keys of all nodes by definition.

To cater for the case where the connection to the CA is broken, Alice could send her current certificate with the puzzle solution in the authentication protocol. An intruder can of course drop the message, but if it does come through, then Bob can first extract Alice's current key and then verify that the message has been sent by Alice. From then on, Alice and Bob use MAC authentication anyway. But this only works if Alice is uncompromised and frequently communicates with Bob. If she does not send Bob any messages or if she is disconnected from Bob for a longer time, an intruder can compromise her old key and then mask as Alice. So although this mechanism may diminish the intruder's possibilities, it cannot guarantee authentication.

The violation of (A4) does not pose specific problems as long as (A5) still holds. If implies that messages may arrive at the receiver out of order, but our algorithms do not rely on a particular message order anyway. Violation of (A4) may also imply that (A5) does not hold anymore.

If (A5) is not correct, the transmission delay information may not be accurate. If the violation goes undetected, it will cause our hardware to filter out valid messages, creating an effect similar to a node suffering from a violation of (A2) or (A3). If it is known that the transmission delay information is not valid, e.g., because (A4) has been violated, then the timestamp filter will be deactivated. This will significantly increase the DoS attack opportunities during client puzzle exchanges.

If (A6) is broken, then valid messages may be filtered out, again creating an effect similar to violation of (A2) or (A3) if the problem is not recognized. In addition, old messages of a node with a fast clock may be replayed by the intruder as soon as the message becomes valid on a node with a slow clock.

If the problem is known, then the clock synchronization will mark the timestamps as invalid. Like with (A5), messages cannot be filtered out by the hardware anymore, so the intruder can replay any messages, not just recent ones. But the software must be able to deal with replayed messages anyway.

To conclude, violation of (A1) is the most critical problem and will cause all authentication to fail almost immediately.

Violations of (A2) and (A3) are also critical and can cause authentication to fail for nodes which are affected by it.

Violations of (A5) and (A6) are minor in comparison as long as they are detected. They will just allow the intruder to bypass the hardware message filters and cause more workload on the nodes. If they are not detected, however, they have the same effect as violation of the connectivity assumption (A2) and (A3). Here, we must decide between quality of service and security: If we want high quality of service and if undetected violation of (A5) and (A6) is highly unlikely, then we should use the timestamp filter. If our ultimate goal is as secure authentication as possible and if the probability of undetected violation of (A5) and (A6) is unacceptably high, then we must not use the timestamp filter and will experience more impact from DoS attacks.

Violation of (A4) is not important as long as (A5) holds, but of course this is harder to achieve without (A4).

We can see that in many cases violation of our assumptions will cause authentication to fail, and unfortunately we need a lot of assumptions. Nevertheless, we believe that our mechanisms are a valid contribution to the field in that the additional assumptions on the synchronicity of the system, namely (A4), (A5), and (A6), are just required by our hardware filtering method and exist solely for the purpose of diminishing the effects of replay attacks. Whether they are used or not is the decision of the system designer. They can easily be removed from the algorithms by not enabling hardware timestamping and by deactivating the timestamp filter. The effect will be the same as when timestamps are not valid.

9 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 substantial hardware support helps to keep DoS attacks at bay and that even with hardware, we cannot fully guarantee the absence of DoS attacks in W_2F. In order to use our timestamp filter, nodes must know a lot about the system like message transmission delays for all paths or the global clock synchronization precision π.

Due to the ad-hoc nature of the network, violations of our system assumptions are possible and in this case we may even get into situations where authentication cannot be guaranteed anymore. The problem even increases with mobile nodes which may be disconnected from the network for a longer period of time. If violations of our clock synchronization and message delay assumptions can occur, we must decide between security and quality of service. Either we use the timestamp filter, accepting that it can corrupt authentication if the assumptions are unknowingly violated, or we do not use it and thereby accept a higher impact of DoS attacks.

In this paper, we proposed one possible way to achieve authentication under denial-of-service attacks using public keys with client puzzles for communication establishment and shared keys for the communication itself. The quality of our algorithms stands and falls with our assumptions, so as a next step, we must see if we can find algorithms which require less assumptions than our current ones while providing us with the same protection against DoS attacks. We must also compute the probability of violating our assumptions and investigate if there are ways to reduce it to zero.

Finally, we must subject our algorithms to a formal analysis to prove their correctness. We will also explore the effectiveness of our denial-of-service defences by using the cost-based framework of Meadows [11].

References

[1] T. Aura, P. Nikander, and J. Leiwo. DOS-resistant authentication with client puzzles. In *Proceedings of the 8th International Workshop on Security Protocols*, LNCS 2133, pages 170–177, Apr. 2000.

[2] CERT. Cert advisory: TCP SYN fboding and IP spoofi ng attacks. Technical Report CA-1996-21, Carnegie Mellon Software Engineering Institute, Sept. 19, 1996. http://www.cert.org/advisories/CA-1996-21.html.

[3] W. Diffi e and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[4] L. Garber. Denial-of-service attacks rip the Internet. *IEEE Computer*, 33(4):12–17, Apr. 2000.

[5] L. Gong, P. Lincoln, and J. Rushby. Byzantine agreement with authentication: Observations and applications in tolerating hybrid and link faults. In *Proceedings Dependable Computing for Critical Applications-5*, pages 139–157, Champaign, IL, Sept. 1995.

[6] M. Horauer, U. Schmid, and K. Schossmaier. NTI: A Network Time Interface M-Module for high-accuracy clock synchronization. In *Proceedings 6th International Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'98)*, pages 1067–1076, Orlando, Florida, March 30 – April 3 1998.

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

[8] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[9] K. H. Leung, K. W. Ma, W. K. Wong, and P. H. W. Leong. FPGA implementation of a microcoded elliptic curve cryptographic processor. In *Proceedings of Field-Programmable Custom Computing Machines (FCCM'00)*, pages 68–76, 2000.

[10] K. Matsuura and H. Imai. Modification of Internet Key Exchange resistant against denial-of-service. In *Pre-Proc. of Internet Workshop 2000 (IWS2000)*, pages 167–174, Feb. 2000.

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

[12] D. L. Mills. NTP timescale and leap seconds. http://www.eecis.udel.edu/~ntp/ntp_spool/html/leap.htm.

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

[14] S. W. Smith and S. Weingart. Building a high-performance, programmable secure coprocessor. *Computer Networks*, 31:831–860, 1999.

[15] D. X. Song and A. Perrig. Advanced and authenticated marking schemes for IP traceback. In *Proceedings IEEE Infocom 2001*, Apr. 22-26, 2001.

[16] G. Tsudik. Message authentication with one-way hash functions. *ACM SIGCOMM Computer Communication Review*, 22(5):29–38, Oct. 1992.

[17] H. Wang, D. Zhang, and K. G. Shin. Detecting SYN flooding attacks. In *IEEE Infocom 2002*, June 23-27, 2002.

[18] Y. Zhang and W. Lee. Intrusion detection in wireless ad-hoc networks. In *Proceedings of the 6th Conference on Mobile Computing and Networking*, pages 275–283, Aug. 6-11, 2000.

[19] L. Zhou. *Towards Fault-Tolerant and Secure On-Line Services*. PhD thesis, Department of Computer Science, Cornell University, Ithaca, NY USA, May 2001.

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

[21] L. Zhou, F. B. Schneider, and R. van Renesse. COCA: A secure distributed on-line certification authority. Technical Report TR2000-1828, Computer Science Department, Cornell University, Dec. 2000.