Technische Universität
Braunschweig

Physikalisch-Technische
Bundesanstalt

Projet Thesis

Analysis of the NTP Autokey Procedures

Stephen Röttger

February 8, 2012

Technische Universität Braunschweig
Institute of Theoretical Information Technology
Prof. Dr. Jiri Adamek

mentored by Dr. Dieter Sibold
and Dr. Stefan Milius

Contents

1. Introduction

The German Energiewirtschaftsgesetz ("Energy Industry Act") states that new buildings shall be equipped with smart meters which give the users information about the actual energy consumption and the actual period of use. In this context, "smart" actually means "able to communicate", i.e. the meters communicate with the operators of the measurement points and can, thus, transmit data concerning the consumption and the periods of use automatically. This is aimed at issuing bills often enough to allow customers to actively control their consumption. Pursuant to the currently valid Verification Ordinance, these meters are subject to metrological verification if their read-outs and their rate switching times are not visible when the housing is closed. This applies in particular to meters that are not equipped with a display for reasons of costs. To guarantee accurate time for these devices, they have to be synchronized with a reliable time source, e.g., a time server of the energy distribution network operator; this process has to be authenticated. Authentication is important in order to rule out tampering by third parties. To this end, the NTP Autokey procedure is investigated in this paper; this procedure is to enable authenticated time synchronization in conjunction with the Network Time Protocol (NTP) [1].

The network time protocol is the most wide-spread time synchronization protocol on the Internet; it provides time synchronization in three different modes. In the *Client-Server Mode*, a client establishes a connection to a server and retrieves the time from this server. In the *symmetric mod*e, two NTP servers adjust their respective time with each other. In the broadcast mode, an NTP server sends its time updates to a broadcast address. For each of these modes, the NTP Autokey procedure described in RFC 5906 [2] provides an extension which is meant to ensure authentication of the NTP server.

The connections between the server and the client are called "associations". An NTP server does not keep any information about its clients. To retrieve the time, a client sends an NTP packet (cf. Fig. 1.1) containing the time t_1 at which it was sent. The server replies to this with a packet containing the arrival time t_2 as well as the sending time t_3 of the reply. Together with the arrival time t_4 of this reply, the client can compute the time offset $\Delta t = \frac{1}{2}\left((t_2 - t_1) - (t_3 - t_4)\right)$ to the server [3].

Optionally, NTP packets can contain one or more of the extension fields

shown in Fig. 1.2. Each extension field contains an opcode – which describes the type of the extension and, thus, the content of the value fields – as well as data of variable length and an optional signature.

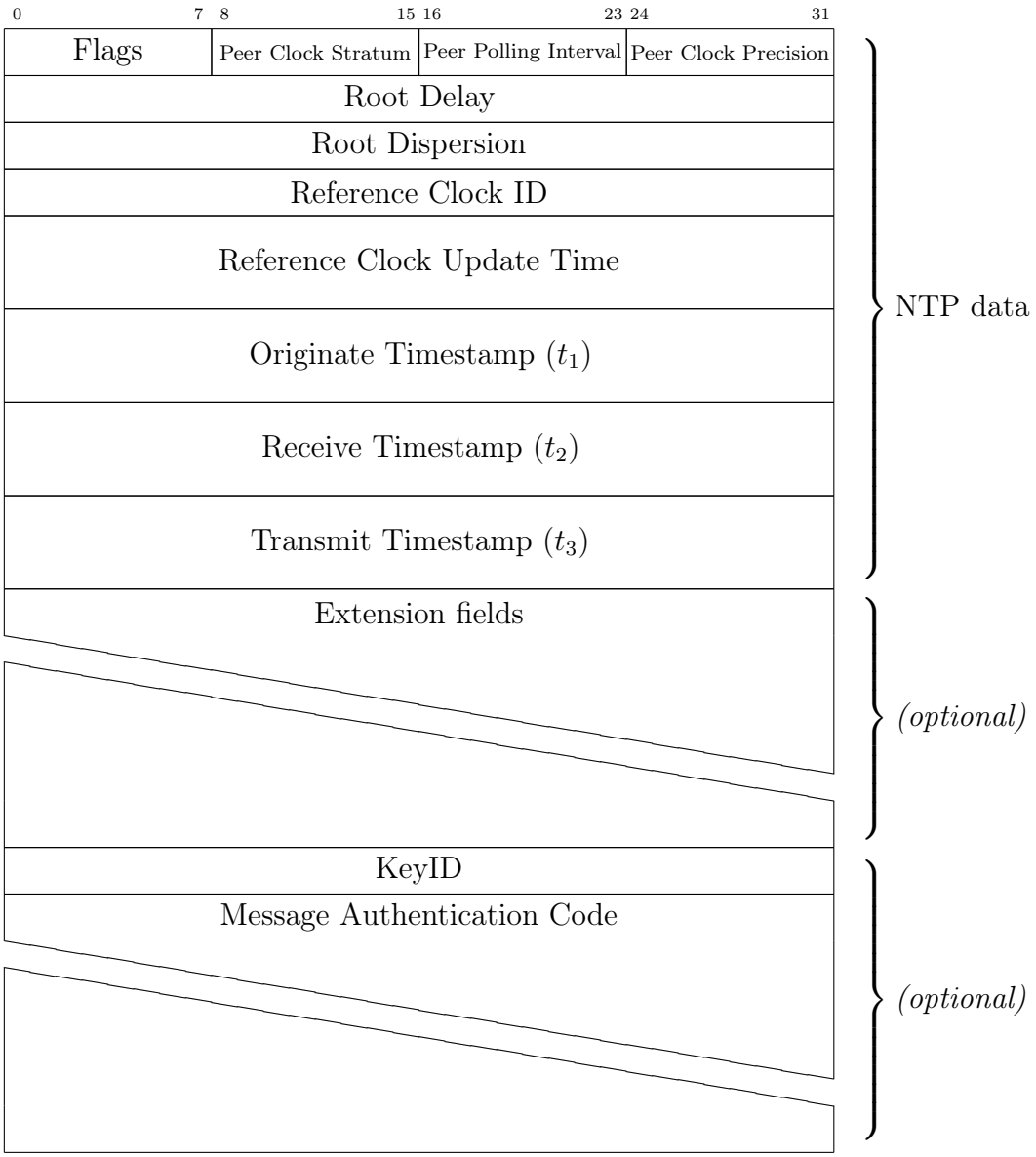

Figure 1.1.: Format of the NTP packet

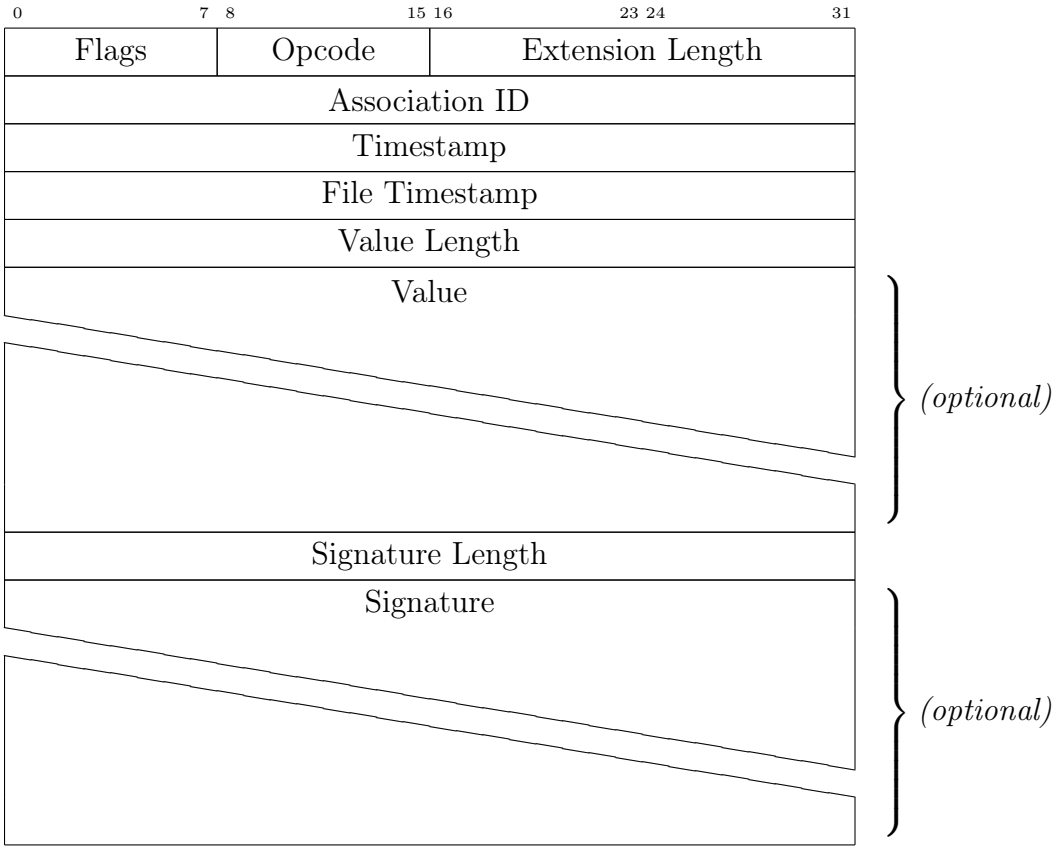

Figure 1.2.: Format of the Extension field

2. Analysis of the NTP Autokey Procedure

The NTP Autokey procedure is based on symmetric keys, the so-called "autokeys"; each of them is valid for one packet only. A 32-bit key ID and a message authentication code (MAC) are attached to each NTP packet. The key ID is a clear identification of the autokey used. The MAC is used to verify the authenticity of the packet. It can only be generated if the autokey matching the key ID is known or if it can be computed.

NTP extension fields are used for the exchange of data which are necessary for the autokey authentication. This occurs at the beginning of the autokey communication and, if necessary, to update the autokey parameters. The MAC for packets containing an NTP extension field is computed exclusively with public values and can, thus, not ensure authenticity. For such packets, the optional signature in the extension field can ensure authenticity; this, however, protects the values contained in the extension field only. The time information contained in such a packet is not secured and can be tampered with by an adversary while remaining unnoticed.

2.1. Client-Server Mode

Within the autokey procedure, the client and the server share a secret called cookie which they are the only ones to know. The server chooses this secret at the beginning of an association and transmits it to the client in a encoded form so that no one else can read it. The cookie is then used to determine the autokey from a key ID. The autokey is computed as follows:[1]

$$\text{Autokey} = H\,(\text{Sender-IP} \,||\, \text{Empfänger-IP} \,||\, \text{KeyID} \,||\, \text{Cookie})\ . \qquad (2.1)$$

$H()$ is a cryptographic hash function. The MD5 procedure is used to compute the autokey, so that the resulting key has a length of 128 bits. The sender and the recipient IP addresses consist of 32 bits each if IPv4 is used or of 128

[1] || designates the concatenation of the data.

bits if IPv6 is used. The key ID and the cookie have a length of 32 bits each. Subsequently, the MAC (Message Authentication Code) for the packet can be computed:

$$\text{MAC} = H\left(\text{Autokey} \mid\mid \text{NTP packet}\right) \tag{2.2}$$

Diverse hash algorithms can be used for the MAC. The reference implementation supports the MD5 and the SHA1 hash algorithms, wherein the algorithm used is recognized by means of the MAC's length. When receiving a packet, the recipient computes the autokey using the cookie, then computes the MAC of the received packet and compares it with the MAC which was attached to the packet by the sender. If the two coincide, then the recipient knows that the sender of the packet is in possession of the cookie. Obviously, it is critical to keep the cookie secret, since an adversary who is in possession of the cookie can issue MACs himself and can, thus, masquerade as the server.

Since the server does not keep a record on its clients, it cannot choose a random cookie but has to be able to re-compute it at each request from a client. In addition, each client has to receive an unequivocal cookie, and third parties must not be able to issue cookies themselves. To this end, the server has a random 32-bit value, the "server seed", which is used to compute cookies and is kept secret. The cookie is computed as:

$$\text{Cookie} = \text{MSBs}_{32}\left(H\left(\text{Client IP} \mid\mid \text{Server IP} \mid\mid 0 \mid\mid \text{Server Seed}\right)\right) . \tag{2.3}$$

$\text{MSBs}_{32}()$ cuts off the 32 most significant bits of the hash function's result. The 0 is composed of 32 bits. The hash function used is MD5. The cookie of a client thus never changes, except if the server changes its server seed. If this happens, the server notices that a client is still using an old cookie, as its requests also contain a MAC. The client's MAC can, thus, not be verified by the server, so that the server replies with a negative acknowledgement (crypto NAK). The client then resets the association to the server and re-negotiates the parameters.

2.1.1. Authenticity Verification

In order for the client to be able to verify the authenticity of the server, extension fields are signed by the server. Thus, the packet with which the server sends the cookie to the client also contains a signature. In order to verify the signature, the client needs the public key of the server and has to be sure that this key belongs to the server and that it can be trusted. To this end, the server sends a certificate to the client. The client can verify its reliability using one of five identity schemes.

Trusted Certificate In the *trusted certificate* scheme, the server sends additional certificates which establish a certificate chain from the server itself to a trusted authority. Contrary to common practice, the client does not store the certificates from trusted authorities but accepts a certificate sent by the server as that of a trusted authority if it contains an "X.509v3 Extended Key Usage" extension field with the content "trustRoot".

Private Certificate Certificate In the *private certificate* scheme, the server and the client have the same certificate with the corresponding private key. This scheme corresponds to a pre-shared key procedure; it guarantees authenticity as long as the private key remains secret. If the same key is used for several clients, each of these clients can masquerade as the server towards the other clients. It is possible to limit this risk by means of NTP secure groups, but this scheme presents absolutely no advantages whatsoever compared with the usage of NTP in combination with symmetric keys [1].

Schnorr Identify Friend or Foe The *Identify Friend or Foe (IFF) Scheme* [2] is a challenge-response procedure using a modified Schnorr signature algorithm [4]. The client is in possession of public parameters of the server; the client knows for sure that these parameters belong to the server. These parameters, however, do not have to be kept secret. The client issues a random value – the challenge – and sends it to the server. The latter can then carry out a computation which proves to the client that it is in possession of one of the secrets which are part of the public parameters. A more accurate description of this procedure and of the following two is given in SectionA.

Guillou Quisquater The *Guillou-Quisquater (GQ) Scheme* [2] is based on a signature procedure invented by L. Guillou and J. Quisquater [5]. In this challenge-response scheme, the client and the server need a shared secret – the so-called "group key". The group key is used to generate the key for the challenge-response procedure and to verify the response (cf. SectionA.2). Similar to the *private certificate* scheme, a client can masquerade as the server towards other clients if the same group key is used for all of them.

Mu Varadharajan The *Mu-Varadharajan (MV) scheme* [2] is a challenge-response protocol which is based on an encryption procedure invented by Y. Mu and V. Varadharajan [6]. The encryption procedure was created to encrypt broadcasts, so that data can be decrypted using several keys; in addition, individual keys can be added or revoked.

In order to use this procedure as an identity scheme, the client sends a random value to the server. This value is encrypted by the server and sent back to the client. The client decrypts the key and compares it to the original value. A detailed description of the procedure is given in Section A.3.

2.1.2. Protocol Sequence

At the beginning of an association, the client and the server exchange data in several steps; the client needs this data to verify the authenticity of the server and of the subsequent time responses. For this, the client sends requests attaching an extension field. The client then receives corresponding responses from the server. Depending on the identity scheme used, the client is then already in possession of public parameters or of a certificate from a trusted authority with which the client can verify the identity of the server. Communication then takes place in the following order:

ASSOC The first request from the client contains an extension field of the type *association*. The request contains a 32-bit status field and the autokey host name of the client. Correspondingly, the server's response contains a status field and the autokey host name of the server. The status field contains information on the cryptographic algorithms used by the server and the client.

CERT Then, the client sends a *certificate* request (except for the *private certificate scheme*). The latter contains a host name for which the client wishes to receive a certificate. In a first step, this is the host name the client had received from the server through the *association* message in the previous step. The response of the server then contains a certificate for the requested host name. If this certificate has not been issued by the server itself, the client also requests the certificate of the issuer. If the client finally receives a self-generated certificate in the response, then it checks whether this is a certificate from a trusted authority. A certificate chain is, thus, established from a trusted authority up to the server with which the client is communicating, so that the server can be trusted. Attention must be paid to the fact that the client recognizes a certificate as being from a trusted authority if "trustedRoot" is contained in an extension field of this certificate. Confidence is not, as usually is the case, established by the fact that the client is already in possession of the certificate from the trusted authority, but by the identity scheme used with the corresponding public parameters.

IFF/GQ/MV If a challenge-response identity scheme is used, the client subsequently sends a packet with an extension field of the corresponding type: *IFF, GQ* or *MV*. This packet contains the client's challenge, and the server's reply contains the response corresponding to the identity scheme used, together with a signature. The parameters that the client needs to verify the response have already been sent to him prior to the startup procedure in a secure way, so that these cannot have been exchanged by an adversary.

COOK In a next step, the client requests the secret cookie from the server. The cookie request hereby contains a public key of the client which can be used for encryption. The server computes the cookie (cf. formula 2.3), encrypts the cookie using the client's public key and sends it in an extension field with a signature. The client verifies the server's signature and decrypts the cookie.

At this point, the client has verified the server's authenticity and received the secret cookie. Now, the client synchronizes its time with the server's by means of regular NTP packets that do not contain an extension field, but an attached key ID and MAC. The client can choose the key ID freely and compute the corresponding autokeys (cf. formula 2.1). The server replies to these packets according to the regular NTP procedure and also attaches a key ID and a MAC to the responses. The key ID used by the server is the key ID from the corresponding request from the client. When computing the corresponding autokeys, however, attention must be paid to the fact that the two IP addresses are inverted, which results in an autokey that is different to that used in the request.

SIGN After the time has been synchronized, the client sends a *signature* request in an extension field. This request contains the client's certificate. The server signs this certificate and sends it back to the client.

Subsequently, the client is in a mode in which only milliseconds are synchronized with the server and considerable time steps are no longer performed. This still takes place using extension-field-free NTP packets with MAC.

2.1.3. Weak Spots/Security Analysis

There are two possible aims for an adversary:

- denial of service, and

- masquerading to tamper with the client's time.

Denial of service (DoS) stands for attacks that are aimed at saturating a computer or a service to cause an overload, so that this computer or server is no longer available. These attacks can often not be prevented if the adversary has enough resources[2] at his disposal; there can, however, also be protocol weak spots that facilitate such attacks even further. In the following, we will only consider masquerading weak spots, since the risk they represent is much higher. In a masquerading attack, an adversary masquerades as an NTP server towards the client and can, thus, tamper with its local time.

Generally, a client sends time requests to several servers at the same time. These requests are identified in the client configurations either by an IP address or by a DNS name. Optionally, the client is in possession of the public parameters of the identity schemes of the servers in order to be able to verify their authenticity. For the analysis, it is presupposed that the adversary does not physically have access to the client and can therefore not tamper with the configuration files or exchange parameters. Since the NTP protocol compares all servers with each other in order to filter out faulty time sources, the adversary has to be able to change the packets of a majority of servers. In the following, it is therefore presupposed that the adversary is located as a "man in the middle", i.e. between the client and a majority of the servers.

To tamper with the client's time, the adversary has to be able to respond to time requests on the part of the client by means of an NTP packet containing a valid MAC or a MAC that is accepted by the client. This can happen if the cryptographic primitives used exhibit security vulnerabilities or if the adversary gets in possession of the private parameters of an identity scheme and can, thus, bypass the identity scheme, or if the adversary gets in possession of the secret cookie which is used for authentication between the client and the server.

Insufficient bit lengths With 32 bits, the server seed from which client cookies are computed is too short. An adversary can connect with a server and request a cookie. The only secret information which are used to compute the cookie is the server seed (cf. formula 2.3). By means of a brute-force attack, the adversary can find out the server seed on the basis of the cookie obtained. For this, he tries out each possible server seed, as described in Algorithm 1, and can then compute his own cookie. If this cookie coincides with that obtained from the server, the adversary has found the correct server seed and can then compute cookies for other clients. These cookies can be used by the adversary to

[2]For instance processing power or network throughput

masquerade as the server towards the corresponding clients, since the adversary is able to forge message authentication codes.

Algorithm 1 Brute-force algorithm to compute the server seed.

```
for i = 0 → 2^32 − 1 do
  C_i ← H (Server-IP || Client-IP || 0 || i)
  if C_i = Cookie then
    return i
  end if
end for
```

This algorithm was tested on an Intel Core i5 with $2 \cdot 2,53\,\text{GHz}$ using a single thread. It was also able to perform the 2^{32} MD5 computations in less than 25 minutes.

With 32 bits, the cookie itself is also too short. The adversary can capture a packet from the client-to-server communication and retrieve the cookie from this packet by means of a brute-force attack. To this end the adversary proceeds as described in algorithm 1. He issues MACs for the captured packet and uses all possible values from 0 to $2^{32} - 1$ as a cookie. He then compares each of the results with the MAC which has been attached to the packet. If these coincide, the adversary has discovered the cookie and can, again, masquerade as the server. For each cookie, the adversary has to perform an MD5 computation in order to obtain the corresponding autokey which he subsequently concatenates with the 48-bit long NTP packet and hashes again. He thus has to perform two MD5 computations for each cookie, i.e. the attack takes about twice as long as the brute-force attack on the server seed.

Client Identity Check Another security vulnerability lies in the fact that the identity of the client is not verified. The server performs no authenticity verifications, but relies on the client's IP address instead. In addition, the server does not have any record about the client and cannot detect a request which deviates from the protocol sequence. This can be exploited by an adversary to obtain a client's cookie. The only prerequisite is that the adversary must be able to spoof his own IP address to that of the client, to receive the responses to this IP address and to forward them to the client or delete them according to his requirements. This is the case in the "man-in-the-middle" scenario.

To obtain the cookie, the adversary changes his IP address to the client's and sends a cookie request to the server. This request contains a public key (the adversary's) which the server uses to encrypt the cookie. Then, the server sends the encrypted cookie back to the client's IP address. The adversary

obtains this packet, since he is located between the server and the client; he can then decrypt the cookie with his private key. Now, he is in possession of the client's key and can, from then on, masquerade as the server. Figure 2.1 shows a sequence diagram of this attack.

Identity schemes The *trusted certificate scheme* provides no security against attacks whatsoever. In this scheme, the client automatically trusts any certificate containing the text "trustRoot" in an extension field. An adversary can easily issue such a certificate and, in this way, masquerade as any possible server.

The three *challenge-response schemes* (IFF, GQ and MV) provide no protection against adversarys either. A "man in the middle" can forward a challenge from the client to the authentic server. He will then cut off the signature from the server's reply which contains the valid response and replace this signature with his own. In this way, the client supposes that the adversary was able to reply to the challenge and thus trusts him. In addition, all three cryptographic protocols exhibit severe security vulnerabilities which make them fully superfluous and always allow a potential adversary to send a response which is accepted by the client. These security vulnerabilities are described in more detail in Section A.

From all identity schemes mentioned in the standard, only the *private certificate scheme* guarantees the authenticity of the server. However, it uses pre-shared keys and, thus, offers no advantage compared with the NTP symmetric key procedure [1]. Only by using NTP secure groups is it possible to attribute a unique key to each client. Otherwise, it would be possible for clients to masquerade as the server towards other clients.

Although RFC 5906 [2] already describes that the trusted certificate scheme provides no security against "man-in-the-middle" attacks, it is not possible to configure the reference implementation, so that this mode is considered insecure. The selection of the identity scheme is determined by the server when the connection is established, even if the client has configured it for another scheme. Also, no warning is generated if the server uses the insecure trusted certificate scheme; the only possibility of detecting it is to read it out manually using the programme "ntpq". There, the 32-bit status word, which is exchanged between the server and the client via the association message, is indicated as a hexadecimal number. Four of these 32 bits identify the identity scheme used.

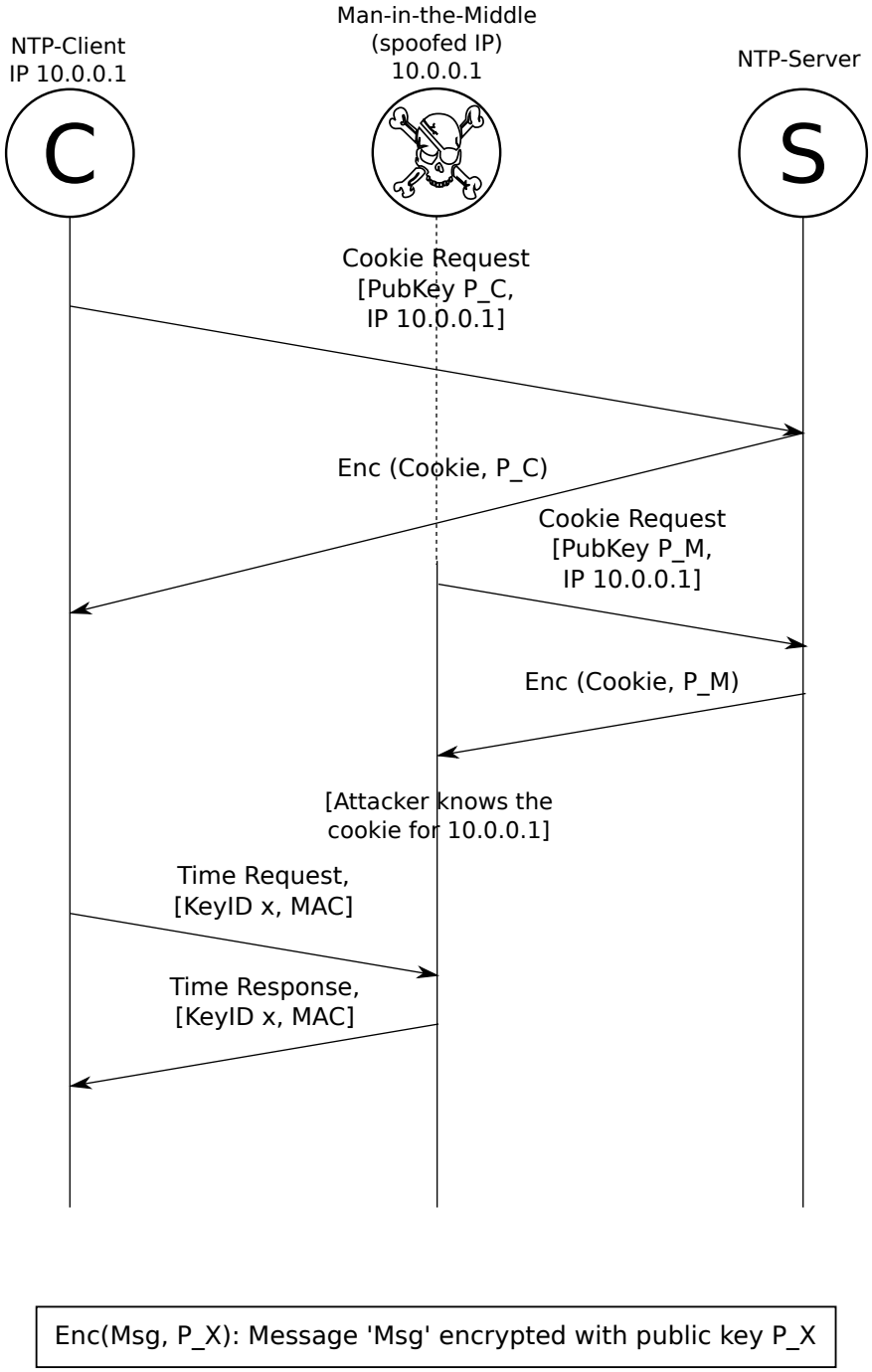

Figure 2.1.: Exploit of the lacking identity check of the clients in the NTP autokey protocol.

2.2. Symmetric Mode

In the symmetric mode, two NTP servers synchronize their time with each other. One hereby differentiates between the active peer, which initiates the connection, and the passive peer. The message authentication codes and the autokeys are computed like in the client-server mode. The difference resides in the computation of the cookies. Whereas in the client-server mode, the cookie is re-computed for each packet, in the symmetric mode, both peers remember the cookie. Hence, one of the peers can select the cookie as a random number.

2.2.1. Key Generation

As an additional security measure, the key IDs are not chosen at random, but are computed according to a one-time password procedure created by L. Lamport [7]. Both peers compute a series of key IDs according to this procedure and transmit the first element of this series to each other in a secure way. Upon receipt of a packet, the other peer can thus check whether the key ID contained in the packet belongs to this series.

To this end, one of the peers selects a random key ID I_0 as a starting value and, using this value and the cookie that has previously been agreed upon, computes the first autokey K_0 according to formula 2.1. He then uses the first 32 bits of K_0 as the next key ID and so forth for all further key IDs I_j and autokeys K_j für $j \in \{1, \ldots, n\}$:

$$K_j = H\,(\text{sender IP} \;||\; \text{recipient IP} \;||\; I_j \;||\; \text{cookie})$$

$$I_j = \text{MSBs}_{32}\,(K_{j-1})$$

These autokeys and key IDs are now used in the reverse order. The peer first sends its partner I_n in a secure way. For the message authentication codes of the subsequent packets, it uses I_{n-1}, I_{n-2}, $\ldots$, I_0 in a row as Key IDs. The partner remembers the key ID I_x of the last packet received from the peer. Upon receipt of a new packet with the key ID I_y, it computes $I_z = \text{MSBs}_{32}\,(H\,(\text{sender IP} \;||\; \text{recipient IP} \;||\; I_y \;||\; \text{cookie}))$ and accepts the packet only if $I_z = I_x$ is valid.

This procedure is designed to prevent a third party from computing – without huge computational effort – key IDs itself which would then be accepted by the communication partner. For this, the third party would have to be able to break the preimage resistance[3] of the cryptographic hash function used.

[3]For a given hash h, it should be practically impossible to find a message M to which $\text{H}\,(M) = h$ applies.

This procedure is, however, vulnerable to a brute-force attack, since with their 32 bits, the key IDs used are too short.

The disadvantage of this procedure is that the first 32 bits of each key are revealed, since they correlate with the previously used key ID. The effective key length hence decreases from 128 bits to 96 bits.

In the client-server mode (cf. Section 2.1), the client also generates its key IDs using this procedure. In this case, however, this does not lead to increased security, since the server does not keep a record on the client and does not remember the key ID received last. Since only the client uses this procedure, at best the server could verify the key ID of the request. The critical part – residing in the fact that the client verifies the key IDs of the server – is not ensured since the server simply uses the key ID of the request in its replies.

2.2.2. Protocol Sequence

In the symmetric mode, the exchange of parameters runs bi-directionally. Each response packet from a peer can simultaneously contain also an own request. In the following, one differentiates between the active peer, which initiates the connection, and the passive peer.

1. The connection is initiated by means of an association packet, similar to the client-server mode. As described in Section 2.1.2, the association packet contains the autokey host name of the peer as well as a status field with supported cryptographic algorithms. The passive peer's response also works similarly to the client-server mode.

2. Depending on the identity scheme used, certificates are then exchanged and a challenge-response protocol is carried out. This takes place bi-directionally. The sequence is the same as in the client-server mode, except that each peer once plays the role of the server and once the role of the client.

3. The peer that has first carried out the certificate exchange or the challenge-response protocol as the client sends a cookie request to its communication partner. The request contains the public key of the sender's which has to be used for encryption; the response contains the encrypted cookie which will subsequently be used as a shared secret. Contrary to the client-server mode, the cookie is chosen at random and saved by both peers.

4. As described in Section 2.2.1, both peers generate a list of autokeys for themselves and request from the other peer the last element of this

list with an extension field of the type "autokey". The autokey request contains no data, whereas the reply contains the key ID of the last element on the list as well as the corresponding index.

5. From then on, both peers can synchronize their time by sending normal NTP packets with an attached key ID and MAC. Upon receipt of such a packet, the recipient checks whether the key ID fits in the list by hashing it several times and by comparing the result with that received in the previous step (cf. Section 2.2.1); eventually, it computes the corresponding autokey by means of which the message authentication code attached to the packet is verified.

2.2.3. Weak Spots/Security Analysis

The symmetric mode has, just like the client-server mode, certain weak spots which an adversary can exploit to bypass the whole authentication procedure. The generation of keys according to Lamport's one-time password procedure only protects against the generation of packets by an adversary, but not against modification. An adversary as a "man in the middle" must hence capture replies from a peer and modify them; he cannot generate his own responses. Since the key IDs are, however, only 32 bits long, he can – if he happens to be in possession of the cookie – compute valid key IDs by means of a brute-force attack in a maximum of 2^{32} steps.

Contrary to the client-server mode, it is, in the symmetric mode, not possible for the adversary to request the cookie from a peer by spoofing its IP address, since the cookie is generated as a random value. The peer can either detect that the request occurs outside the protocol order and ignore it, or generate a new cookie which deviates from that which is in possession of the other peer.

Similarly, a brute-force attack on the server seed is no longer possible, since the server seed is no longer used for generating the cookie. The brute-force attack on the cookie as described in Section 2.1.3 is, however, still possible since it is only 32 bits long. Furthermore, the problems of the identity schemes remain. This can allow the adversary to bypass the whole authentication procedure and masquerade as an arbitrary host if he captures the establishment of the connection.

3. Alternatives

At present, there is, apart from the autokey procedure, no standardized procedure of authenticated time synchronization. Since the autokey procedure is, however, faulty and does not provide any security, the only alternative – apart from revising the autokey procedure – is to use an existing time synchronization procedure and to perform authentication at a lower layer of the network stack.

Authentication, when it does not take place in the application protocol, yields a series of disadvantages for time synchronization, and these lead to inaccuracy. This inaccuracy is caused by the fact that the return travel paths of time requests are increased unequally by the authentication protocols. In its error correction, the NTP procedure presupposes that both travel paths take the same time, since it knows only the sum of the two. The halved difference of these times thus leads to inaccuracy.

- Protocols that ensure authentication usually perform an exchange of keys at the beginning. The time it takes to exchange keys increases the time a time request needs on its way out.
- Since the application does not know about and does not control the authentication procedure, the times required for signatures cannot be compensated for. If the authentication is implemented inside the application protocol, the time it takes to generate the signature is estimated and compensated for by dating ahead the time-stamp in the packet.
- The authentication mechanisms of a third protocol are not designed for the requirements of time synchronization. Time-consuming algorithms are frequently used for the signature, and most of the protocols additionally encrypt the data.
- Authentication protocols are often stateful. Hence, a permanent connection must be either maintained between the server and the client, or established again for each packet. In the latter case, a new parameter negotiation takes place for each packet. This increases the time of outward travel.

In the following, diverse protocols, which can be used to ensure the authentication of NTP communication, will be considered.

3.1. IPsec

IPsec is a protocol on top of the link layer that ensures encryption and authentication by means of two independent headers. The authentication header (AH) protects the authenticity of the payload by means of a MAC or of a signature, whereas the encapsulating security payload additionally provides privacy by means of encryption. An overview of the protocol is given in RFC 4301 [8]; the authentication header and the encapsulating security payload are specified in RFC 4302 [9] and RFC 4303 [10], resp. IPsec can be operated in two different modes. The transport mode establishes an end-to-end connection between two computers and, for this, inserts the IPsec header between the IP header and the header of the transport layer protocol. In the tunnel mode, in contrast, the connection is secured over part of the communication path. To this end, the IP packet as a whole is packed into another IP packet including the IPsec header; this packet is then sent to the tunnel end point. The latter retrieves the original IP packet and forwards it.

Using IPsec for the authentication of NTP packets has the advantage that the application need not be modified. In addition, encryption is optional and, thus, causes no superfluous overhead. On the other hand, being located on the link layer represents a disadvantage compared with the other protocols. It requires an integration into the network stack of the operating system which is normally implemented in the kernel. In common operating systems, IPsec is, thus, also part of the kernel, whereas protocols on higher network layers can be implemented via libraries in the user mode and, in this way, do not affect the operating system. N. Fergusson and B. Schneier analyzed IPsec in a paper in 2003 [11] in which they detected a certain number of vulnerabilities. They came to the conclusion that IPsec is currently the best existing protocol for securing IP packets; however, its high complexity can easily lead to insecure configurations, so that its use in security-critical areas is not recommended. The complexity of the exchange of keys was, however, reduced by a new version of the Internet key exchange protocol (IKE) [12] in 2005.

3.2. TLS Tunnel (e.g. Open VPN)

Transport Layer Security (TLS) is the downward-compatible successor of Secure Sockets Layers (SSL). The protocol is described in RFC 5246 [13] and provides the encryption and authentication of the data beyond the transport layer. A TLS tunnel, e.g. Open VPN, sends IP packets via a TLS connection. At the end point of the TLS tunnel, these packets are then decrypted, their authenticity is verified, and they are forwarded to their destination.

As TLS acts on top of the transport layer, the protocol can be made available via a library in the user mode. In addition, a TLS tunnel, however, requires the support of the operating system for virtual network interfaces which are used as the end point for the TLS connection. Since, as in the case of IPsec, whole IP packets are transmitted, the encryption and authentication are transparent for the application and they need not be modified. In addition, TLS is an established standard which has often been checked for security vulnerabilities; it is therefore considered secure [14, 15, 16].

3.3. DTLS

Datagram Transport Layer Security (DTLS) is an adaption of the TLS protocol to stateless transport-layer protocols such as the User Datagram Protocol (UDP) or the Datagram Congestion Control Protocol (DCCP). DTLS was developed in 2004 [17] and published as RFC 4347 [18] in 2006, using UDP as a transport-layer protocol. An important design criterion was to modify the TLS protocol as little as possible in order not to impair its security properties. It was, however, necessary to implement reliability[1] for handshakes; sequence and epoch numbers were introduced by means of which the loss of messages for a new parameter exchange as well as replays of packets can be detected.

Using DTLS has the advantage that it can be implemented totally within the application by using a library and, thus, does not affect the operating system at all. This, however, is also a disadvantage, since the application then has to be modified to use DTLS. Since the protocol is comparatively young, no security analysis can be found in literature. Due to its great similarity to TLS, it is, however, relatively improbable that it exhibits security vulnerabilities which do not already exist in TLS itself.

[1] A connection is considered *reliable* if packet losses are compensated for and if it is ensured that the packets are ordered.

3.4. Conclusions

The authentication of the NTP data can be implemented either on the transport layer, or in the form of a tunnel for IP packets. Transport-layer protocols have the advantage of not affecting the operating system; on the other hand, they require an integration into the application. A modification of the application, however, increases the maintenance effort, since new versions – which can, for instance, contain critical security updates – require the modifications made to be adapted. The transport-layer protocol DTLS has, to date, not been analyzed as to its security; however, since it is based on the TLS protocol – which is considered secure – it can be taken into consideration as an alternative. Due to the modifications it requires, it can, however, not be precluded that DTLS brings about security vulnerabilities that do not exist in TLS.

IP tunnels, in contrast, are established by the operating system and are transparent for the application. Hence, the application need not be modified. When comparing IPsec and TLS-based tunnel solutions directly, TLS is clearly preferable. In the case of IPsec, although mainly its complexity has been rated negatively, several security analyses of TLS could not detect any critical security vulnerabilities. In addition, IPsec is normally implemented in the kernel of the operating system, which may prevent access to the source code for analysis and causes us to be bound to the implementation of the manufacturer of the operating system.

4. NTP Autokey Improvements

As shown in Section 2.1.3, the autokey protocol exhibits a series of critical security vulnerabilities which make the protocol inefficient and cannot guarantee authenticity. In this section, modifications to the autokey protocol are presented which mitigate the identified security vulnerabilities:

- Since the identity schemes, as demonstrated earlier, cannot guarantee authenticity, they are replaced by a hierarchic public key infrastructure based on X.509 certificates.
- The client's public key used for encryption is involved in the computation of the cookie. This prevents a potential adversary from obtaining a client's cookie from the server, since the cookie is encrypted with the client's public key when it is sent and can, thus, not be decrypted by the adversary. However, if the adversary sends the key request with his own public key, then the cookie he receives from the server differs from that of the client.
- In order to avoid brute-force attacks, the lengths of the cookie and of the server seed are increased from 32 to 128 bits.
- The signature contained in extension fields covers the whole NTP packet.
- *Optionally*: The computing of the message authentication code is replaced by a keyed-hash message authentication code (HMAC).

4.1. Hash and MAC Algorithms

The message authentication code is computed after the transmit time-stamp t_3 has been added to the packet by the server. The time it takes to compute the MAC is therefore part of the return path delays and causes an asymmetry between the path delays from client to server and back, which, in turn, results in the inaccuracy of the synchronization. It is conceivable to forward date the transmit time-stamp and to wait until the recorded time has been reached after computing the MAC; for this, however, the time of the MAC computation would

have to be estimated, and depending on the server load, it can vary considerably. Nevertheless, this proceeding can compensate for the time inaccuracy caused by the computing of the MAC when the server load is normal. Hence, the selection of the utilized hash and MAC algorithms primarily affects the load of the server, whereas it affects the accuracy only under certain circumstances.

In the following, the security of possible algorithms will be analyzed. Two requirements are placed on a cryptologic hash function H. It has to be a *one-way function*. This means that it should be practically impossible to find a plain text M for a hash value h so that $H(M) = h$ applies. This property is also called "preimage resistance". In addition, the hash function should be collision-resistant. This means that it should be practically impossible to find two plain texts M_1 and M_2 to which $H(M_1) = H(M_2)$ applies.

In the NTP autokey procedure, a hash function is used in three places. The MD5 procedure is used to compute the cookie and the autokeys. To compute the MAC, the server can choose from the SHA-1 and the MD5 procedures. It was shown by means of attacks that these two procedures are not collision-resistant. For SHA-1, a collision can be found with a complexity of 2^{69} steps [19], whereas 2^{21} steps suffice in the case of MD5 [20]. Attacks on the preimage resistance are, however, not known for any of the two procedures.

For the computing of the cookie and the autokey, collisions do not play any role, since the corresponding plain text cannot be chosen by an adversary. An attack on the preimage resistance could possibly cause the cookie from an autokey or the server seed from the cookie to be revealed. For the computing of the MAC, however, a successful attack on the collision resistance would cause a server to be able to generate two packets that would have the same MAC. Since the server can generate MACs for any packets anyway, this does not seem to open up any possibility for attacks. Collision resistance is a critical property when a server signs data from a third party, since these data can have been chosen in such a way that the signature is valid also for other data. Stevens et al. [21], for example, have demonstrated that, by attacking the collision resistance of MD5, it was possible to have a valid certificate-authority certificate issued. Attacking the preimage resistance could, in contrast, possibly cause the autokey from a MAC to be revealed. Hence, only the preimage resistance of the hash function used is critical, whereas the collision resistance is negligible.

As shown previously, MD5 suffices as a hash function in all areas of the autokey protocol. Since however, the hash function used does not have any influence on the synchronization accuracy, it makes sense to choose an algorithm which is recommended by the German Federal Office for Information Security (BSI) or NIST. For this purpose, the Federal Information Processing Standard (FIPS) 140-2 [22] is available; it lists cryptographic algorithms that have been

validated by NIST, and also contains an overview of suitable algorithms provided by the BSI [23]. SHA-256 is part of it.

Message Authentication Code Algorithms for generating Message Authentication Codes are often based on hash functions or block cipher. The algorithm used in the autokey protocol uses the hash value from the concatenation of the autokey and the NTP packet as a MAC. This construction is generally not a good idea. If the hash function used is based on the Merkle-Damgård construction[1] [24], as is the case for, e.g., MD5 and the SHA algorithms, an adversary can capture an authenticated packet, attach data of his own to it and compute the new MAC as a hash of the new data, hereby using the old MAC as the initialization vector for the hash function. It may happen that the adversary additionally has to add a padding used by the hash function to the packet. In the NTP protocol, however, this attack is not possible, since the packets in which the MAC is used for authentication always have the same length. Yet it makes sense to replace the MAC algorithm by an algorithm proposed in FIPS 140-2, e.g., a keyed-hash message authentication code (HMAC) [25] with SHA-256.

It is also conceivable not to prescribe the hash and MAC algorithms in the protocol but to leave this choice to the server. To this end, the algorithms used could be coded in the association extension field in which 4 bits have become available due to the suppression of the identity schemes. The server could then use less demanding algorithms — provided conformity with FIPS 140-2 is not a prerequisite.

4.2. Authentication

By extending the autokey protocol, each NTP packet containing an extension field contains a signature. This signature, however, is computed over the whole packet instead of being computed over the data of the extension field only, as described earlier. NTP packets without extension fields, which are used for time synchronization, use a MAC for authentication, as is the case in the original protocol. To compute this MAC, a secret shared by the server and the client is used, namely the cookie which the server can re-compute at each request of the client. The cookie C is computed by the server from the server seed S and from a public key P_C of the client which is used for encryption.

[1] In the case of the Merkle Damgård construction, blocks are processed sequentially, and the issuing of the last block is used as entry for the processing of the next one.

The server seed is a 128-bit value which is chosen randomly and is kept secret by the server. The cookie is then computed as shown below:

$$C = H\left(S, P_C\right),$$

where H is a hash function selected according to the considerations described in Section 4.1. In addition, the result of the hash function should have a minimum length of 128 bits, since the cookie could be obtained from a captured NTP packet in the event of a brute-force attack if the values are too small – as described in Section 2.1.3. Since the server does not keep a record on its clients and, thus, has to re-compute the cookie at each request, the client has to send its public key along with each request. The autokeys A_I used to compute the MAC are now computed from a public keyID I which is 32 bits long and is also attached to the packets sent.

$$A_I = H\left(I, C\right)$$

The length of the autokey, again, depends on the hash function used and should be at least 128 bits long.

4.3. Protocol Sequence

Association Message Similar to the original autokey protocol, the cycle starts with a packet containing an extension field of the type association. In the request, this extension field contains the host name of the client as well as a status word which contains the algorithms used for signatures as well as the status of the connection. Correspondingly, the response contains the host name of the server as well as the signature algorithms. The bits which had been written by the identity scheme used are now left out and can optionally be used to indicate the hash and MAC algorithms used.

Certificate Message In a next step, the client requests a certificate chain from the server by first requesting a certificate for the server's host name and then for the originator of the certificate obtained, until the client finally receives a certificate that has been issued by a trusted authority. The client recognizes a trusted authority, since it is already in possession of certificates which are recognized as TAs. In addition, the client has to check whether the certificates obtained are also authorized to issue new certificates. To this end, the client tests whether the X.509 extension contains "CA:TRUE". The responses of the

server each contain an X.509 certificate for the host name requested by the client.

If a valid certificate chain to a TA has been established, then the client can verify the server's signatures, and hence, the authenticity of the messages containing an extension field is ensured.

Cookie Message In a next step, the client requests the server's cookie which is used to generate the autokeys. To this end, the cookie request contains the client's public key. The server computes the cookie, encrypts it using the client's public key received and sends it back to the client.

Time request message The modified autokey protocol requires a new extension field. Extension fields of the type *time request* contain the client's public key which has been used to compute the cookie and which the server requires to generate a MAC. This extension field, however, only occurs in the request. The reply is a normal NTP packet without an extension field to which a keyID and a MAC have been attached.

4.4. Security considerations

The authenticity of all packets from the server is protected by the modifications of the autokey protocol. NTP packets which contain an extension field are protected by means of signatures whose key is traceable to a trusted authority via a certificate chain. The X.509 certificates used for this purpose have a limited period of validity. This causes the certificates to become invalid as soon as their hash or signature algorithms or the length of the keys used no longer offer sufficient security due to progress in research and in processing power. Since the client generally has no reliable time at its disposal when verifying the certificates, it is impossible to check whether a certificate has already expired. Hence, the client has to decide which algorithms and key lengths are admissible and ensure sufficient security. Another important mechanism of PKIs is the possibility of revoking certificates. In the case of X.509 certificates, this is usually implemented via the Online Certificate Status Protocol (OCSP) or via Certificate Revocation Lists (CRL). For OCSP, a URL is stored in the certificate; the client can request the validity of the certificate from this URL. One of these procedures also has to be implemented by NTP clients, since a certificate issued to an unauthorized person by mistake would otherwise be valid indefinitely.

The authenticity of packets without an extension field is based on the security of the MAC algorithm used as well as on the secrecy of the corresponding autokey or of the cookie from which it has been generated. The secrecy of the cookie, in turn, is guaranteed, since it is issued using the client's public key and is encrypted with this key for transmission, whereas the authenticity of the cookie is protected by a signature, as described above.

Brute-force attacks are avoided since the server seed, the cookie and the autokeys have a sufficient length (128 bits). Replay attacks are likewise impossible, since packets from the server always contain the time at which the corresponding request was sent by the client and the client only accepts packets with regard to which a request has been issued.

Since NTP packets contain the arrival and sending time, timing attacks are facilitated. Timing attacks measure the time that cryptographic computations have taken and draw conclusions on the key used [26]. When implementing the signature algorithms, counter measures should be integrated in order to avoid timing attacks. If, for instance, the RSA implementation of the OpenSSL library is used, this is already the case by means of RSA blinding [26].

If RSA is used as a signature algorithm, attention must be paid to additionally using a secure padding scheme such as, e.g., the Optimal Asymmetric Encryption Padding (OAEP) [27]. Otherwise (when using flawed padding schemes), the information as to whether a ciphertext corresponds to the scheme upon decryption can allow an adversary to decrypt messages [28].

Despite authentication, the eventuality of an attack in which the adversary delays the return path of the NTP packets remains. When computing the time difference between the client and the server, the NTP protocol presupposes that the delay of the packets on the way out and the way back are roughly the same. The halved difference between the two paths results in an inaccuracy. This attack is, however, mitigated by the fact that the NTP reference implementation dismisses packets which took more than 16seconds round trip delay. An adversary could, thus, modify the time of a client by max. 8 seconds.

5. Conclusions

The NTP autokey procedure has the task of authenticating the time synchronization of the NTP protocol and, in this way, of preventing disturbances caused by third parties. The analysis performed has, however, revealed several security vulnerabilities which make the whole procedure inefficient. Insufficient bit lengths are used for secret values, so that they can be discovered by an adversary by means of trial and error; the identity schemes which are supposed to guarantee the authenticity of the server's certificate provide no security against a "man-in-the-middle" attack, and last but not least, the client's identity is not verified, so that an adversary can request the secret cookie from the server for any client. This makes it possible for an adversary to masquerade as any server towards an NTP client and to transmit a wrong time to this client.

Since no other protocol exists at present to perform an authenticated time synchronization, alternatives have been investigated which can protect the NTP packets on a lower network layer by means of authentication. These alternatives, however, have the disadvantage of reducing time synchronization accuracy. Comparing IPsec, TLS and DTLS, using a TLS IP tunnel is recommended as a transitional solution.

In the long run, though, an authentication in the application protocol is necessary to enable higher synchronization accuracy. To this end, certain modifications to the NTP autokey procedure have been suggested in order to mitigate the security vulnerabilities revealed. In addition, the security requirements placed on the hash and MAC algorithms used have been investigated. The results have shown that the MD5 procedure and the MAC algorithm used in the original autokey procedure guarantee sufficient security.

Bibliography

[1] Mills, D., J. Martin, J. Burbank, and W. Kasch: *Network Time Protocol Version 4: Protocol and Algorithms Specification.* RFC 5905 (Proposed Standard), June 2010.

[2] Haberman, B. and D. Mills: *Network Time Protocol Version 4: Autokey Specification.* RFC 5906 (Informational), June 2010.

[3] Mills, David L.: *Computer Network Time Synchronization: The Network Time Protocol.* CRC Press, Inc., 2006.

[4] Schnorr, C. P.: *Efficient signature generation by smart cards.* Journal of Cryptology, 4:161–174, 1991.

[5] Guillou, L. C. and J. J Quisquater: *A "paradoxical" identity-based signature scheme resulting from zero-knowledge.* In *International Crytology Conference*, 1989.

[6] Mu, Yi and Vijay Varadharajan: *Robust and secure broadcasting.* In Rangan, C. and Cunsheng Ding (editors): *Progress in Cryptology — INDOCRYPT 2001*, volume 2247 of *Lecture Notes in Computer Science*, pages 223–231. Springer Berlin / Heidelberg, 2001.

[7] Lamport, Leslie: *Password authentication with insecure communication.* Communications of the ACM, 24:770–772, November 1981.

[8] Kent, S. and K. Seo: *Security Architecture for the Internet Protocol.* RFC 4301 (Proposed Standard), December 2005. Updated by RFC 6040.

[9] Kent, S.: *IP Authentication Header.* RFC 4302 (Proposed Standard), December 2005.

[10] Kent, S.: *IP Encapsulating Security Payload (ESP).* RFC 4303 (Proposed Standard), December 2005.

[11] Ferguson, Niels and Bruce Schneier: *A cryptographic evaluation of ipsec.* Technical report, Counterpane Internet Security, Inc, 2000.

[12] Kaufman, C., P. Hoffman, Y. Nir, and P. Eronen: *Internet Key Exchange Protocol Version 2 (IKEv2)*. RFC 5996 (Proposed Standard), September 2010.

[13] Dierks, T. and E. Rescorla: *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246, Network Working Group, August 2008.

[14] Paulson, Lawrence C.: *Inductive analysis of the internet protocol tls*. ACM Transactions on Information and System Security, 2:332–351, 1997.

[15] Mitchell, John C., Vitaly Shmatikov, and Ulrich Stern: *Finite-state analysis of ssl 3.0*. In *In Seventh USENIX Security Symposium*, pages 201–216, 1998.

[16] Wagner, David and Bruce Schneier: *Analysis of the ssl 3.0 protocol*. In *In Proceedings Of The Second Unix Workshop On Electronic Commerce*, pages 29–40. USENIX Association, 1996.

[17] Modadugu, Nagendra and Eric Rescorla: *The design and implementation of datagram tls*. In *In Proc. NDSS*, 2004.

[18] Rescorla, E. and N. Modadugu: *Datagram Transport Layer Security*. RFC 4347 (Proposed Standard), April 2006.

[19] Wang, Xiaoyun, Yiqun Lisa Yin, and Hongbo Yu: *Finding collisions in the full sha-1*. In *In Proceedings of Crypto*, pages 17–36. Springer, 2005.

[20] Xie, Tao and Dengguo Feng: *How to find weak input differences for md5 collision attacks*. Cryptology ePrint Archive, Report 2009/223, 2009.

[21] Stevens, Marc, Er Sotirov, David Molnar, Dag Arne Osvik, and Benne De Weger: *chosenprefix collisions for md5 and the creation of a rogue ca certificate*. In *http://eprint.iacr.org/2009/111, Crypto 2009, Springer-Verlag 2009*.

[22] National Institute of Standards and Technology: *Fips pub 140-2, security requirements for cryptographic modules, annex a: Approved security functions*. Technical report, Department of Commerce, January 2011.

[23] Bundesnetzagentur für Elektrizität, Gas, Telekommunikation, Post und Eisenbahnen: *Bekanntmachung zur elektronischen signatur nach dem signaturgesetz und der signaturverordnung (Übersicht über geeignete algorithmen)*. Technical report, June 2011.

[24] Merkle, Ralph Charles: *Secrecy, Authentication and Public Key Systems.* PhD thesis, Stanford University, June 1979.

[25] Krawczyk, H., M. Bellare, and R. Canetti: *HMAC: Keyed-Hashing for Message Authentication.* RFC 2104 (Informational), February 1997.

[26] Brumley, David and Dan Boneh: *Remote timing attacks are practical.* In *In Proceedings of the 12th USENIX Security Symposium*, pages 1–14, 2003.

[27] Bellare, Mihir and Phillip Rogaway: *Optimal asymmetric encryption*, 1994.

[28] Bleichenbacher, Daniel: *Chosen ciphertext attacks against protocols based on the rsa encryption standard pkcs1.* Springer-Verlag, 1998.

A. Challenge-response Identity Schemes

A.1. Schnorr Identify Friend or Foe

The IFF challenge-response protocol [2, 3] is based on a signature procedure by C. Schnorr [4]. To this end, the server or a trusted authority (TA) generates a 512-bit prime number p, a 160-bit prime number q to which $q|(p-1)$ applies, a $g \in \mathbb{Z}_p$ of the order[1] q as well as a random private group key $b \in \mathbb{Z}_q \setminus \{0\}$ and the corresponding public key $v = g^{q-b} \mod p$. To analyse a response from the server, the client needs (p, q, g, v). These do not necessarily have to be kept secret, but the client should obtain them in a way that guarantees that the parameters belong to the server and cannot have been exchanged by an adversary.

Protocol Sequence

1. The client selects a random number $r \in \mathbb{Z}_q \setminus \{0\}$ as a challenge and sends it to the server.
2. The server selects a random number $k \in \mathbb{Z}_q \setminus \{0\}$, computes $y = k + b \cdot r \mod q$ and $x = g^k \mod p$ and sends $(y, H(x))$ as a response to the client[2].
3. The client tests the response by computing $z = g^y \cdot v^r \mod p$ and verifying $H(z) = H(x)$.

$H(z) = H(x)$ applies, since:

$$\begin{aligned} z &= g^y \cdot v^r \mod p \\ &= g^{k+b\cdot r} \cdot g^{r\cdot(q-b)} \mod p \\ &= g^k \cdot g^{b\cdot r+q\cdot r-b\cdot r} \mod p \\ &= g^k \mod p = x \end{aligned}$$

[1] g is of order q in $\mathbb{Z}_p \Leftrightarrow g^q \equiv_p 1 \wedge \forall x \in \mathbb{Z}_q : g^x \not\equiv_p 1$.
[2] $H(x)$ is a cryptographic hash function.

Weak spots This challenge-response protocol can be outwitted by an adversary if the latter sends $(0, H(v^r \mod p))$ to the client in the 2nd step. The client then computes $z = g^0 \cdot v^r \mod p = v^r \mod p$ and, thus, compares $H(v^r \mod p) = H(v^r \mod p)$ and accepts the response.

A.2. Guillou-Quisquater

The Guillou-Quisquater challenge-response protocol [2, 3] is based on a signature procedure invented by L. Guillou and J. Quisquater [5]. The server and the client are hereby in possession of a shared key which has to be kept secret. The server or a TA generates two large prime numbers p and q, whose product $n = p \cdot q$ has a length of 512 bits, and another large prime number as a group key $b \in \mathbb{Z}_n \setminus \{0\}$. Subsequently, (n, b) is distributed to all clients using a secure way. These parameters must not be captured by a third party, since anyone in possession of these parameters can masquerade as the server. In addition, the server generates a random private key $u \in \mathbb{Z}_n \setminus \{0\}$ and the corresponding public key $v = (u^{-1})^b \mod n$.

Protocol Sequence

1. The client sends a random $r \in \mathbb{Z}_n \setminus \{0\}$ to the server as a challenge.
2. The server selects a random $k \in \mathbb{Z}_n \setminus \{0\}$, computes $y = k \cdot u^r \mod n$ and $x = k^b \mod n$ and sends $(y, H(x))$ to the client.
3. The client computes $z = v^r \cdot y^b \mod n$ and accepts the response provided $H(z) = H(x)$ applies.

$H(z) = H(x)$ applies, since:

$$\begin{aligned} z &= v^r \cdot y^b \mod n \\ &= \left(\left(u^{-1}\right)^b\right)^r \cdot (k \cdot u^r)^b \mod n \\ &= u^{-b \cdot r} \cdot u^{b \cdot r} \cdot k^b \mod n \\ &= k^b \mod n = x \end{aligned}$$

Weak spots This challenge-response protocol can also be bypassed by an adversary. If the adversary sends $(1, H(v^r))$ in the 2nd step, the client's computation yields $z = v^r \cdot 1^b$. The client then accepts the response as it compares $H(z)$ mit $H(v^r)$.

A.3. Mu-Varadharajan

The Mu-Varadharajan challenge-response protocol [2, 3] is based on an encryption procedure invented by Y. Mu and V. Varadharajan [6]. The original procedure is dedicated to encrypting broadcasts to several clients. For any client, keys can be added or deleted arbitrarily without the other clients needing new keys.

Key Generation

The following key generation was left out in RFC 5906 [2] and described in detail in the book "Computer Network Time Synchronization: The Network Time Protocol" [3] instead.

The server or a TA generates a key E, partial keys $\bar{g}$ and $\hat{g}$, as well as a key $(\bar{x}_j, \hat{x}_j)$ for each client j. The keys are computed in accordance with $E^{-1} \equiv_p \bar{g}^{\hat{x}_j} \cdot \hat{g}^{\bar{x}_j}$ for each client j.

1. To this end the server generates n different m-bit primes $s'_j,\ j \in \{1 \dots n\}$ and $q = \prod_{j=1}^{n} s'_j$, so that $p = 2 \cdot q + 1$ is a prime and selects a generator g from $\mathbb{Z}_p^*$.

2. For each s'_j the server computes $s_j = \frac{q+s'_j}{s'_j}$; thus $s_j \cdot s'_j \equiv_q s'_j$ is valid.

3. The server selects n different numbers $x_j \in \mathbb{Z}_q \backslash \{0\}$ with $\gcd(x_j, q) = 1$ and constructs a polynominal of the form $\prod_{j=1}^{n}(x - x_j)$. The polynominal will be expanded in the form $\sum_{i=0}^{n} a_i \cdot x^i$; for each a_i a $g_i = g^{a_i}$ will be calculated.

4. Now, the so-called "master encryption key" $A = \prod_{i=1}^{n} \prod_{j=1}^{n} g_i^{x_j}$ can be calculated.

5. Out of $\mathbb{Z}_q^*$, a private group key b will be chosen arbitrarily.

6. The client's keys are calculated according to $\bar{x}_j = b^{-1} \cdot (\sum_{i=1}^{n} x_i^n) - b^{-1} \cdot x_j^n \mod q$ and $\hat{x}_j = s_j \cdot x_j^n \mod q$.

7. For the generation of the activation key $s = \frac{q}{s'_j}$, a key (the s'_j used) has to be revoked.

8. Now, the key E and the corresponding partial keys $\bar{g}$ and $\hat{g}$ can be calculated according to $E = A^s$, $\bar{g} = \bar{x}^s \mod p$, and $\hat{g} = \hat{x}^{s \cdot b} \mod p$.

The clients receive $(p, \bar{x}_j, \hat{x}_j)$ by secure means. The server needs $(p, E, \bar{g}, \hat{g})$ to be able to reply to challenges.

Protocol Sequence

1. The client sends a random number $r \in \mathbb{Z}_q \setminus \{0\}$ as a challenge to the server.

2. The server selects an arbitrary $k \in \mathbb{Z}_q \setminus \{0\}$ and calculates $E' = E^k \mod p$, $\bar{g}' = \bar{g}^k \mod p$ along with $\hat{g}' = \hat{g}^k \mod p$. After calculating $x = E' \cdot r \mod p$, it sends $(x, \bar{g}', \hat{g}')$ to the client.

3. The client calculates $E'^{-1} = \bar{g}'^{\hat{x}_j} \cdot \hat{g}'^{\bar{x}_j}$, encrypts x and accepts the response if $E'^{-1} \cdot x \equiv_p r$ applies.

This protocol sequence is in accordance with RFC 5906. The protocol sequence in the mentioned book [3] differs from that, because there in the 2nd step $H(x)$ instead of x is sent to the client. Hence, the client cannot encrypt x to compare it to r but has to invert E'^{-1} in order to obtain E'. Thereupon, he compares the $H(x)$ obtained with $H\left(E' \cdot r \mod p\right)$ and accepts the response if they coincide. The NTP reference implementation, however, conforms to the sequence described in the RFC.

Weak Spots

An adversary who wants to masquerade as a legitimate server can send $(r, 1, 1)$ to the client in the 2nd step of the protocol. The client then computes $E' = \left(1^{\hat{x}_j} \cdot 1^{\bar{x}_j}\right)^{-1} = 1$, $z = E' \cdot r = r$ and compares z to the r received. Since these are equal, the client accepts the adversary's response.

Furthermore, the adversary has the possibility to retrieve a valid combination of E', $\bar{g}'$ and $\hat{g}'$ if he knows q. This is, for example, the case if he himself is one of the server's clients. For this, he sends a request to the server with a random r and, from the server's response, computes $E' = x \cdot r^{-1} \mod p$. Subsequently, the adversary can reply to requests of other clients himself without being in possession of the original key E.

In the protocol sequence according to the book [3], it is not possible to perform the second attack, since x is forwarded in a hashed form only. However, if the adversary is in possession of any client parameter, he is nonetheless able to calculate E' according to $E' = \left(\bar{g}'^{\hat{x}_j} \cdot \hat{g}'^{\bar{x}_j}\right)^{-1}$ and, as a consequence, is in possession of a valid combination of E', $\bar{g}'$ and $\hat{g}'$. Consequently each client can masquerade as a server towards another client.

B. Contents of the CD

In the following, a short overview of the contents of the data storage medium enclosed is givenn:

/ausarbeitung/ contains this paper as .tex source and as .pdf

images/ contains the images used as .svg vector graphics

/dokumente/ contains the documents quoted in the present paper

/poc/ contains the Proof-of-Concept code which can bypass the autokey procedure in the event of a “man-in-the-middle” attack