# Simple PKI

Sebastian Wiesner
Supervisor: Ralph Holz
Seminar Innovative Internet Technologies and Mobile Communications
Chair for Network Architectures and Services
Fakultät für Informatik, Technische Universität München
Email: basti.wiesner@mytum.de

## ABSTRACT

In this paper we discuss the SPKI standard as an alternative to the current X.509 and OpenPGP standards. The paper starts with a short history of PKI, and assesses the current state and the various flaws in the X.509 and OpenPGP standards. Then the main part of this paper explains the concepts of SPKI, and discusses how SPKI supports various notions of trust. Finally the paper concludes with an attempt to determine why SPKI lacks widespread implementation and deployment despite its advantages over conventional PKI.

## Keywords

OpenPGP, PKI, SPKI, Trust, X.509

## 1. A SHORT HISTORY OF PKI

In their famous 1976 paper “New Directions in Cryptography” [10] Whitfield Diffie and Martin Hellman not only invented public key cryptography, they also envisioned the first PKI: “The enciphering key E can be made public by placing it in a public directory along with the user's name and address” [10, page 648].

Two years later the term “certificate” was introduced to describe cryptographically signed bindings from public keys to names that were to be used independently of the *public file* suggested by Diffie and Hellman [19].

By 1988 these ideas had developed into the X.509 standard, part of the X.500 family of standards. Following the idea of Diffie and Hellman these standards envision a “global, distributed directory of named entities” [14, page 6]. X.509 certificates consequently bind public keys to *global names*. These certificates are issued by *certificate authorities*.

In 1991 Phil Zimmerman published PGP (standardized as OpenPGP [5]), an alternative certificate infrastructure without certificate authorities. Instead OpenPGP allows entities and individuals to issue their own certificates. By mutual signing of certificates among different entitities, OpenPGP forms a “web of trust” [2, 8] in which people assert the validity of each others certificates. The idea behind this system is that given enough signatures all signatures of a certificate can jointly assert the validity of the certificates, even though some individual signatures might potentially be fraudulent.

In 1998, SPKI [13, 14] joined the family of public-key cryptography standards, following ideas of Ron Rivest [27] and Carl M. Ellison [12]. On the surface it can be considered as a combination of the hierarchical X.509 infrastructure and the flat PGP web. However, in fact SPKI abandons some basic assumptions of X.509 and PGP: The need of global names for identification, and the need of identification itself. SPKI uses *local names* that live within name spaces, and it introduces *authorization certificates* to allow for authorization without the need of a confirmed identity.

Today X.509 is used standards such as S/MIME [24] and SSL/TLS [9], making it the dominant certification infrastructure for secure HTTP communication and confidential mail communication. However, of late several serious incidents concerning certification authorities have raised concerns about the security of the current CA infrastructure. For instance, in 2011 over 200 fraudulent certificates were issued and supposedly used in MITM attacks after a breach into the Dutch DigiNotar CA [3, 15, 21, 22] resulting in the removal of DigiNotar from the root stores of major browsers and the subsequent bankruptcy of DigiNotar.

OpenPGP is widely used to ensure authentication and integrity of the infrastructure of free software projects[1], mostly because certificates can be issued and used without fees or additional costs, since no certification authorities are involved in the issuing of certificates and all necessary tools are provided free of charge[2]. However, OpenPGP has not been widely adopted among individuals, and thus there is no large web of trust as originally indented.

SPKI, however, has not been adopted at all. The purpose of this paper is to introduce SPKI, to assess in how far it is an improvement over X.509 and PGP, and to discuss why it was not adopted by software vendors and the internet community. The paper starts with a short introduction into the current PKI situations, its problems and its flaws. The following main section introduces SPKI, explaining the core concepts of name spaces and certificates. The next section determines how a notion of trust can be represented in SPKI, and compares this to the trust notion in OpenPGP and X.509. The paper concludes with a discussion of SPKI's success (or the lack thereof).

[1]For instance, the Debian project uses PGP to sign the packages in its software repository in order to guarantee their authenticity and integrity.
[2]For instance, the Free Software Foundation provides a fully featured PGP suite with its GNU Privacy Guard (GnuPG or PGP) project under a free software license with no charge.

## 2. PKI—THE CURRENT STATE

Nowadays X.509 and OpenPGP are the dominating standards for public-key cryptography. On the surface, both standards appear to be vastly different, in fact even antithetical, however it turns out that these standards share some very basic assumptions on what a PKI is supposed to provide and how it should work.

### 2.1 The problem of global names

The most basic assumption of OpenPGP and X.509 is that names are *global*. This assumption is inherited from X.500 vision of a single global directory tree of named entities. In this directory tree, each entity is represented by a specific node, and this node is address by a globally unique *distinguished name*. Originally X.509 certificates were merely access tokens, granting a keyholder the permission to modify the corresponding node in the directory tree. Certificate authorities were not independent entities, but associated with nodes in the directory tree, issuing access certificates for subordinate nodes [14].

Only after it became clear that X.500 would never be deployed, X.509 certificates were extracted from the X.500 family and and—following the business needs—used as *identity certificates*. With the lack of a single directory tree, there was also no single certificate hierarchy any longer. Instead many independent and competing CAs continued to issue certificates for arbitrary global names. By and large the X.509 as used today thus ignores and even contradicts most of its original design concepts. Peter Gutmann gives and intriguing overview of the current X.509 flaws in [16].

OpenPGP [5] is not different in this aspect. Though it does not rely on CAs to issue certificates, the names—called *user ID*—in OpenPGP certificates are still global in the whole OpenPGP network.

However, global names contradict the human use of names. While there are global names—mostly world-wide brands and trademarks like Coca Cola, Apple or Amazon—the *exact meaning* of names that we use depends on our local domain.

For instance, to some people the name *Coca Cola* might refer to the Coca Cola company as a whole, to others it might only refer to the soft drink that goes by this name, and to others again it might stand as a name for all soft drinks of this kind (e.g. Pepsi Cola). Sometimes the meaning of name even depends on the context of its use.

The problem of the *meaning of a name* is increased by the fact that many names that we use are not actually unique. According to the U.S. Social Security Administration the name "Emily" was given to not less than 223,488 female children born in the U.S. between 2000 to 2009 [4]. Thus the name "Emily Smith" is not unique within U.S., probably not even within a larger city. In his popular book "Beyond fear" the renowned cryptographer and security expert Bruce Schneier tells the anecdote that another Bruce Schneier from Illinois is tired of getting Bruce Schneier's e-mail by mistake [28, page 184]. We obviously even fail to correctly identify a person who is present in the public by his books, talks and interviews.

To issue an identity certificate for Emily Smith, the ambiguous local name "Emily Smith" has to be turned into a unique global name artificially. This effectively turns the problem of distributing a public key into the problem of distributing the public name one goes by [16].

The impact of the problem can be observed in OpenPGP. Here keys are identified by user IDs which is typically the e-mail address. However, an e-mail address does not provide information about its owner, and hence cannot identified the person behind the address. Moreover due to its distributed nature OpenPGP cannot enforce unique user IDs: Two different keys may be bound to the same mail address. Thus a mail address cannot sufficiently identify a key. To address this problem, OpenPGP users typically distribute the IDs of their keys. A Key ID is a short and unique representation of the *public key itself*. Thus the problem of securely distributing one's public key is not at all solved by OpenPGP. It is merely turned into the problem of securely distributing the key ID.

### 2.2 Authentication and Authorization

With the trust in global names comes the believe that authenticating a key-holder by its certificate is sufficient to authorize her. The consequence is that authorization *without* authentication is impossible in OpenPGP and X.509. Neither of these standards provides means for anonymous authorization.

Thus any security system built on these standards is forced to identify and authenticate its clients in order to authorize actions, even if the identity of the client is not actually relevant for authorization. However, as explained in the previous section secure identification and authentication are no solved problems which obviously has an adverse impact on the design of any security system that could in theory work securely without any authentication at all.

## 3. SPKI—THE FUTURE?

Following these insights SPKI (Simple Public Key Infrastructure [13]) considers global names a fundamental conceptual flaw both in PGP and X.509 [14]. Instead, SPKI uses the *keys* themselves as global identifiers [14].These are naturally globally unique and hence serve well as a reliable global identifier.

Consequently, *keys* are the principal items in SPKI, contrary to X.509 and OpenPGP which use *names* as principals. This neatly avoids all the problems with names discussed above. Most notably it allows for anonymous authorization: Not names, but keys are the subject of authorization.

### 3.1 Local names

However, keys are hardly tractable by humans. Thus SPKI allows for *local* naming of keys within a *name space*. Each public key has an associated name space. In this name space, keys or names can be assigned with identifiers by *basic names*:

**Definition 3.1.** A *basic name* consists of a public key $K$ and an identifier (being a word over a chosen alphabet) [7, 14].

**Example 3.1.** A *basic name* might be $K$ Fred. $K$ is a key defining the *name space*, Fred an arbitrary identifier that may refer to another key or another basic or compound name.

A basic name is unique within a name space. However, a basic name may refer to multiple other keys or names to express *group memberships*.

A basic name is only meaningful within its name space. There is no immediate relation between names from different name spaces.

**Example 3.2.** $K_1$ Fred and $K_2$ Fred are *different* and *unrelated* local names that may map to different keys, even though both names happen to use the same identifier.

There are no rules restricting the identifiers of local names. Each name space may choose its own, arbitrary set of identifiers. A name space concerned with DNS (e.g. a list of known SSH hosts) may use DNS names as identifiers, another name space used for e-mail encryption may use e-mail addresses, and again another name space created from a local address book may use person names as identifiers.

**Example 3.3.** The following are valid, but again unrelated local names:

- $K_1$ Fred
- $K_2$ fred@example.com
- $K_3$ example.com

There is no top level name space, and not even any kind of hierarchy between name spaces. A priori each name space is unrelated to other name spaces.

## 3.2 Linkage of local namespaces

However, name spaces can be linked by combining local names to form *compound names* [7, 14]:

**Definition 3.2.** A *compound name* is a local name consisting of two or more identifiers.

**Example 3.4.** $K$ Fred Sister is a compound name. $K$ is a key defining the name space again. Within this name space, Fred identifies another arbitrary but fixed $K_F$. In the name space of $K_F$ the identifier Sister maps to yet another arbitrary but fixed key $K_S$ eventually. Hence the name spaces $K$ and $K_F$ are linked with this compound name so that Sister from $K_F$ can be referred to from $K$.

By chaining basic names into compound names one can refer to names from other names spaces. However, compound names too are not meaningful outside of their name space.

**Example 3.5.** $K_1$ Fred Sister and $K_2$ Fred Sister are different and unrelated compound names that may map to different keys.

When no distinction between these two kinds of names is required one may speak of a *local name* or just a *name* to refer to either a *basic name* or a *compound name*.

## 3.3 SPKI certificates

To define a name within the name space of a key SPKI provides *name certificates*. Furthermore SPKI provides *authorization certificates* to delegate authorization to a key or a name. In the SPKI language the term "certificate" is often abbreviated as "cert" [7]. This abbreviation will be used in the following.

### *3.3.1 Name space definitions with name certificates*

Name certs define a basic name within the name space of the signing key. A name cert for the local name $KA$ is a signed[3] tuple of four elements $\mathrm{S}_K(K, A, S, V)$ [7, 14].

**Issuer** $K$ The key $K$ which issued the certificate.

**Identifier** $A$ The identifier of the *basic name* that is defined by this cert. Note that name certs *only* define basic names. Compound names are created by the composition of several names, but never defined directly.

**Subject** $S$ The "meaning" of the basic name $K\,A$, typically the key $K'$ to which the new basic name shall map.

**Validity** $V$ The validity specification. Typically this is a pair $(t_1, t_2)$ meaning that the cert is valid only within the interval $[t_1, t_2]$. However, SPKI also allows for online validity checks. Section 5 of [14] discusses the various validity conditions that may be imposed upon SPKI certificates.

**Example 3.6.** The name cert $\mathrm{S}_K(K, \text{Fred}, K_F, ())$ defines the name $K$ Fred as Fred's key $K_F$ (see example 3.1). The cert has unlimited validity.

As said basic names are not required to be unique. Hence one may issue multiple certs for the same basic name:

**Example 3.7.** The name certs $(K, \text{Friends}, K\,\text{Fred}, ())$ and $(K, \text{Friends}, K\,\text{Alice}, ())$ define the name $K$ Friends both as $K$ Fred and as $K$ Alice respectively. Essentially this defines a group $K$ Friends with the members $K$ Fred and $K$ Alice. Separate name certs are required to define the names $K$ Fred and $K$ Alice.

While name certs may only define local names the subject of a name certificate may also be a compound name:

**Example 3.8.** The cert $(K, \text{Friends}, K\,\text{Fred Sister}, ())$ defines $K$ Friends as the compound name $K$ Fred Sister (see example 3.4). To fully define this compound name both the owner of $K$ and $K$ Fred need to issue further name certs, for instance:

- $(K, \text{Fred}, K_F, ())$ to define $K$ Fred as key $K_F$, that is as the name space of $K_F$.

[3]More precisely it is signed with the private key corresponding to the public key $K$. For the sake of brevity I say "signed by a key $K$" instead of "signed by the private key corresponding to the public key $K$".

- $(K_F, \text{Sister}, K_F\, Alice, ())$ to define $K_F$ Sister as the name $K_F$ Alice. Transitively $K$ Fred Sister is now also defined as $K_F$ Alice.
- $(K_F, \text{Alice}, K_A, ())$ to eventually define $K_F$ Alice as Alice' key $K_A$.

These bindings effectively make $K_F$ Alice a member of the group $K$ Friends. Note however that the owner of $K$ has no authority over names outside of her own name space. Hence the meaning of the name $K$ Fred Sister depends on certificates issued by the owner of $K$ *and* $K$ Fred.

### 3.3.2 Authorization certificates

Names as defined by name certs provide a convenient level of abstraction from keys. However security decisions are typically not made based on *who* someone is (*identity*) but rather on *what* she is allowed to do (*authorization*).

SPKI strictly separates identity established by name certs from authorization which is conveyed by *authorization certificates* or "auth certs". An auth cert is a signed tuple of five elements $\mathrm{S}_K(K, S, d, T, V)$ [7, 14].

**Issuer** $K$ Like with name certs, $K$ is the key that issued this cert. The owner of this $K$ is granting the authorization conveyed by this cert.

**Subject** $S$ A local name or a key that receives the authorization[4].

**Delegation bit** $d$ If set the subject may delegate this authorization or a subset thereof. Delegation is performed by issuing a *new* auth cert to another subject, signed with a key that corresponds to the subject $S$, either directly if $S$ is a key or—if $S$ is a name—indirectly with a key that is defined for $S$ by a name cert. Section 3.3.3 explains the interpretation of this flag in more detail.

**Authorization tag** $T$ The authorization that is granted by this cert. The interpretation and meaning of the authorization depends on the application. The verifier has to implement the appropriate logic to interpret the authorization. SPKI however provides a standard semantic for authorization tags.

**Validity specification** $V$ The validity of the cert, just like for name certs (see section 3.3.1).

**Example 3.9.** The auth cert

$$(K, K_F, 0, read \text{ ftp://example.com/}, ())$$

grants $K_f$ the undelegatable and unlimited authorization to read from the URL `ftp://example.com/`. The owner of $K_F$ can include this authorization certificate in her FTP request to the server and then sign the request with $K_F$. This *proves* the FTP server that the request is legitimate and authorized.

[4]SPKI also has *threshold subjects* to reflect the requirement of having $K$ out of $N$ key holders agree on an authorization to be granted. The discussion of these subjects is beyond the scope of this paper. See section 6.3.3 of [14] and section 10 of [7] for information about threshold subjects.

### 3.3.3 Chains of certificates

Auth certs are not required to grant authorization directly to a specific key. They may also grant authorization to *names*:

**Example 3.10.** The auth cert

$$(K, K\, Fred, 0, read \text{ ftp://example.com/}, ())$$

grants the authorization from example 3.9, but not to a key $K_F$, but to the name $K\, Fred$.

Granting authorization to names has some advantages over granting to keys directly:

- It abstracts from the key, making the authorization resilient in face of key changes. If $Fred$ needs to change her key the auth cert does not the to be re-issued.
- It allows for authorization to be delegated to *groups* as in example 3.7.
- It provides a user-friendly way to refer to the entity that is the subject of a cert, i.e. speaking of $Fred$ is just easier than speaking of a public key $K_F$.

This imposes a difficulty to $K\, Fred$: Since the auth cert does not contain a key to sign the FTP request, he needs a *another* name cert that provides a key $K_F$ for the name $K\, Fred$. She then needs to include the auth cert *and* the name cert into her FTP request to prove her authorization. In short he needs to provide the server a *certificate chain* from the signing key to an auth cert accepted by the server.

The discovery of such chains is an interesting computational problem discussed in [7]. This paper introduces the concept of *rewrite rules* of the form $L \to R$ which rewrite names in certificates.

**Definition 3.3.** Name and auth certs can be represented as rewrite rules:

- A name cert $C = (K, A, S, V)$ is represented as rule $K\, A \to S$.
- An auth cert $C = (K, S, d, T, V)$ is represented as rule $K \boxed{1} \to S \boxed{z}$ with $z = 1$ if $d$ is set and $z = 0$ otherwise. The boxed suffixes (*tickets*) control the delegation of authorization during re-writing, and ensure that undelegatable auth certs are not rewritten any further.

Based on rewrite rules the *composition of certs* is defined:

**Definition 3.4.** Given two certs $C_1$ and $C_2$ with the corresponding rewrite rules $L_1 \to R_1$ and $L_2 \to R_2$ respectively and $R_1 = L_2X$ ($L_2$ is a prefix of $R_1$) then $C_3 = C_1 \circ C_2 = L_1 \to R_2X$. $X$ may be empty.

Composing certs allows to combine multiple certs into a single, "virtual" certs that captures the "meaning" of all these certs:

**Example 3.11.** Two name certs

$$C_1 = (K, \text{Friends}, K \,\text{Fred Sister}, V)$$
$$C_2 = (K, \text{Fred}, K_F, V)$$

*together* define $K$ Friends as $K_F$ Sister. With the rewrite rules corresponding to $C_1$ and $C_2$

$$\begin{aligned} C_1 &= K\,\text{Friends} \rightarrow K\,\text{Fred Sister} \\ C_2 &= K\,\text{Fred} \rightarrow K_F \end{aligned}$$

we can now *compute* the virtual certificate that captures this definition:

$$\begin{aligned} C_3 = C_1 \circ C_2 = K\,\text{Friends} &\rightarrow K_F\,\text{Sister} = \\ &= (K, \text{Friends}, K_F\,\text{Sister}, V) \end{aligned}$$

Now the interpretation of the delegation bit becomes clear. If the delegation bit is set, the right hand side of the rule has a "live" ticket $\boxed{1}$. This right hand side can be rewritten further with other auth certs, reflecting the delegation of authority to another subject.

**Example 3.12.** The auth certs

$$\begin{aligned} C_1 &= (K, K_F, 1, A, V) = K\boxed{1} \rightarrow K_F, \boxed{1} \\ C_2 &= (K_F, K_A, 0, A, V) = K_F\boxed{1} \rightarrow K_A\boxed{0} \end{aligned}$$

can be composed to

$$\begin{aligned} C_3 = C_1 \circ C_2 = K\boxed{1} &\rightarrow K_A\boxed{0} = \\ &= (K, K_A, 0, A, V). \end{aligned}$$

The certificate $C_3$ proves the authorization of the owner of $K_A$ to the owner of $K$. Thus using the chain $C_1, C_2$ the authorization $A$ is effectively delegated to the owner of $K_A$.

If the delegation bit is unset however, the right hand side of a rule has a "dead" ticket $\boxed{0}$. Since the left hand side of an auth cert rule always has a live ticket, the left hand side of an auth cert rule cannot be a prefix of a term with a dead ticket. Thus a right hand side with a dead ticket cannot be rewritten by another auth cert, but only by name certs. Hence the authorization cannot be delegated to other subjects.

**Example 3.13.** The authorization conveyed by the auth cert

$$C_1 = (K, K\,\text{Friends}, 0, A, V) = K\boxed{1} \rightarrow K\,\text{Friends}\boxed{0}$$

cannot be delegated any further, because there is no auth cert can be a prefix of the right hand side of this rule. Thus there is no valid chain that can include another auth cert after this cert.

However this cert can still be rewritten with name certs. Consider the name cert

$$C_2 = (K, \text{Friends}, K_F, V) = K\,\text{Friends} \rightarrow K_F.$$

This cert can be composed with $C_1$ to yield

$$\begin{aligned} C_3 = C_1 \circ C_2 = K\boxed{1} &\rightarrow K_F\boxed{0} = \\ &= (K, K_F, 0, A, V). \end{aligned}$$

Hence using the chain $C_1, C_2$ the owner of $K_F$ can prove her authorization to the owner of $K$. Effectively $K_F$ is granted the authorization $A$ for being a member of the *Friends* group.

The composition of certs is transitive, hence there is a transitive closure $C^+$ for a set of certificates $C$. The *name reduction closure* $C^\#$ is a subset of $C^+$ created by only including compositions that *strictly reduce* the right hand side of the composition.

In order to find a chain that proves authentication $A$ for the key $K$ one now takes the set $C$ of all valid auth certs for $A$ and all valid name certs and computes the name reduction closure $C^\#$. From this closure which contains all intermediate compositions from the auth certs to the key $K$ (if there are any) one can create a graph with a vertex for each key and an edge for each auth cert in $C^\#$.

A certificate chain that proves authorization $A$ for the key $K$ can now be created by a simple breadth-first search over this graph. If no chain is found, no such chain exists in the set $C$ meaning that the owner of $K$ actually lacks the authorization $A$ [7].

## 3.4 S-expressions

SPKI uses S-expressions—as known from the LISP family of languages[5]—to encode names, authentications and even certs[6] These expressions are a human-readable and well-understood syntax to encode data structures which makes SPKI certificates pleasingly simple to read and comprehend if compared to the complexity of the ASN.1-encoded X.509 certificates.

### *3.4.1 Names as S-expressions*

S-expressions starting with the tag `name` encode local names:

**Example 3.14.** Within a cert issued by the key $K$ the S-expression `(name Fred)` encodes the basic name $K$ Fred (see example 3.1). Outside of a cert fully qualified names [14] such as `(name (hash sha1 H(K)) Fred)` must be used to explicitly specify the name space. $H(K)$ must be substituted with the SHA1 hash of the key $K$[7].

**Example 3.15.** Within a cert issued by the key $K$ the S-expression `(name Fred Sister)` encodes the compound name $K$ Fred Sister (see example 3.4). Outside of a cert a fully qualified name must be used, just like in the previous example.

### *3.4.2 Authentication as S-expressions*

The encoding of names as S-expressions is merely convenient but in the encoding of authentication S-expressions become truly powerful. Authorization is encoded as a (nested) list of strings [14] starting with `tag`:

**Example 3.16.** The authorization tag

```
(tag (ftp ftp.example.com /foo/bar))
```

might allow FTP access to the directory `/foo/bar` on the host `ftp.example.com`.

[5]See Rivest's S-expressions page [26] and guide [25] for more information about S-expressions.
[6]The encoding of certs as S-expressions is beyond the scope of this paper. [11] has examples of certs as S-expressions. This S-expression encoding of certs is not mandatory. Section 6.5 of RFC 2693 [14] defines translation rules from X.509 and PGP certificates to SPKI certs.
[7]Of course, other hash algorithms may be used just as well.

SPKI leaves the definition of the meaning of authorization to the application that verifies the authorization, allowing for arbitrarily complex authorizations to be conveyed. However SPKI specifies a simple and entirely optional semantic to interpret and intersect authorizations to provide developers with a standardized, yet flexible interpretation of authorization [14]:

In this semantics each item in an authorization further restricts the granted authorization:

**Example 3.17.** The two authorization tags

```
(tag (ftp ftp.example.com))
(tag (ftp ftp.example.com /foo/bar))
```

intersect to

```
(tag (ftp ftp.example.com /foo/bar))
```

Furthermore some wild card constructs are supported, most notably `(*)`, which matches everything, and `(* set)`, `(* prefix)` and `(* range)` which match against a set of values, a string prefix and a range of values respectively [14].

**Example 3.18.** The two authorization tagss

```
(tag (ftp ftp.example.com
          (* set read write)))
(tag (ftp ftp.example.com
          (* set read delete)))
```

intersect to

```
(tag (ftp ftp.example.com (* set read)))
```

**Example 3.19.** The two authorization tagss

```
(tag (ftp ftp.example.com
          (* prefix /foo/)))
(tag (ftp ftp.example.com
          (* prefix /foo/bar/)))
```

intersect to

```
(tag (ftp ftp.example.com
          (* prefix /foo/bar/)))
```

By relying on this standard semantics developers can easily implement authorization checks in applications. One just needs to construct the complete S-expression that authorizes access to a desired resource and intersect this expression with the expression contained in the authorization certificate that desires access. Access will be granted if the expressions intersect.

## 4. TRUST

SPKI provides a distributed certificate infrastructure. In such an infrastructure participating entities are likely to have only an indirect relation between each other. For instance, in PGP one may receive a key not directly from the key owner, but instead from a 3rd party like a public key server. This key may be signed by other keys whose owners by not be known at all. In SPKI, one may have to verify an authorization delegated to an unknown entity.

Naturally the question arises in how far such signatures or such delegated authorizations can be *trusted* in various aspects. In PGP one might want to know if a signer really verified the key owner's identity before signing the key. In SPKI one might want to know whether the delegator of an authorization really ensured that the subject of the delegation will use the authorization appropriately.

### 4.1 The problem of transitive trust

These questions can be generalized to the question in how far statements of entities are trusted. If the entity making a statement is directly known one can directly assess trust into this entity. One can gather enough information and execute sufficient checks to ensure that this entity will behave as expected.

If the entity is not directly known, this is not longer possible. Instead one has to rely on *somebody else's* statement about the trust into this entity. Of course this statement is affected by the trust one has into the entity making this statement. Trust becomes a *transitive* relation which arises the requirement to adequately communicate trust and to make automated decisions about trust.

### 4.2 Trust in X.509

As explained in section 2.1 there are different competing CAs that issue certificates. Hence every X.509 application has a *root store* containing certificates of implicitly trusted CAs.

However, there is no standardized formal process for the inclusion of CAs in root stores. Every application and every organization has its own processes and guidelines regarding the inclusion of CAs in root stores or their removal thereof. These processes and guidelines greatly vary in quality, and consequently many applications include a lot of CAs in their root stores.

All of these root CAs have equal signing authority and may issue certificates for arbitrary names. They may even create subordinate CAs by issuing *intermediate certificates* which are normally permitted to issue arbitrary certificates themselves.

Thus one effectively trusts a lot of CAs with equal signing authority, many of which are not even known to the user. The security of the whole infrastructure is thus lowered to the weakest CAs reachable via the root store. The trust model of X.509 is simply *unlimited* and *ultimate transitive* trust.

### 4.3 Trust in GPG

OpenPGP provides a far less transitive, but still simple trust model. A OpenPGP user may assign an *Owner trust* level to a key in her key ring. This trust level should reflect in how far the owner of that key is trusted to *introduce* keys, i.e. in how far the owner's signature on another key is trusted. PGP knows three trust levels *not trusted*, *marginally trusted* and *completely trusted* [2, 8, 18].

To OpenPGP these levels assess the quality of a signature of this key. PGP calculates the authenticity of a key as a weighed sum of the number of *marginally trusted* and *completely trusted* signatures on that key, based on the user's preferences on how many marginally or completely trusted signatures make a key trusted.

While these trust levels define the trust given to introduced keys, the trust levels themselves are private and not communicated to the outside. They are *not* transitive.

## 4.4 Trust in SPKI—By example

Contrary to X.509 and OpenPGP SPKI does not specify any kind of trust management model. Trust management in SPKI is left to the application which may implement arbitrary trust semantics and algebras.

One such algebra is presented in [18]. It assesses trust based on a framework called *Subjective Logic* which is essentially a calculus for *opinions* [18]:

**Definition 4.1.** An opinion is a triple $\omega = \{b, d, u\}$ where $b + d + u = 1$ and $\{b, d, u\} \in [0, 1]^3$. The components are the *belief* $b$, the *disbelief* $d$ and the *uncertainty* $u$. $\omega_p^A = \{b_p^A, d_p^A, u_p^A\}$ is the opinion of an *agent* $A$ about a statement $p$.

The *belief* and *disbelief* components describe the probability of whether a statement is true or not. The *uncertainty* component compensates for the absence of knowledge about a statement.

Such opinions provide the base for a logic calculus of *conjunction*, *recommendation* and *consensus* [18][8].:

**Definition 4.2.** The *conjunction* $\omega_{p\wedge q}^A = \omega_p^A \wedge \omega_q^A$ of the opinions $\omega_p^A$ and $\omega_q^A$ about two discrete binary statements $p$ and $q$ represents $A$'s opinion about $p$ *and* $q$ being true.

**Definition 4.3.** The *recommendation* $\omega_p^{AB} = \omega_B^A \otimes \omega_p^B$ is $A$'s opinion about the statement $p$ *as a result of a recommendation from* $B$. $\omega_B^A$ is $A$'s opinion about $B$'s recommendations. $\omega_p^B$ is $B$'s opinion about the statement $p$ *as recommended to* $A$[9].

In order two assess trust in an SPKI certificate two special opinions are of importance [18]:

**Definition 4.4.** The opinion $\omega_{KA(K_I)}^R$ is the opinion of the recipient $R$ of a certificate about the *authenticity of the key* $K_I$ of the certificate's issuer $I$, that is, whether the key really belongs to the issuer.

**Definition 4.5.** The opinion $\omega_{RT(I)}^R$ is the opinion about the recipient $R$ of a certificate about the *recommendation trustworthiness* of the certificate's issuer $I$, that is, how much $R$ trusts $I$ to issue certificates correctly[10].

[8]The exact mathematical definition of these operators are not reproduced here, as these are not relevant for the discussion of this algebra within the scope of this paper. Refer to section 3 of [18] for details.

[9]This is not necessarily identical to $B$'S *real* opinion about $p$.

[10]This is essentially equivalent to the Owner trust of PGP, however at a much finer level.

Together these two opinions form $R$'s opinion about a certificate issued by $I$, expressed as the *conjunctive recommendation term* $\omega_I^R = \omega_{KA(K_I)}^R \wedge \omega_{RT(I)}^R$ [18]. The trust in a certificate subject $S$ is consequently expressed as a recommendation $\omega_S^R = \omega_I^R \otimes \omega_{KA(K_S)}^I$ where $K_S$ is the public key of $S$. By chaining such expressions one can now compute the relative trust of a certificate chain in which each certificate's subject has an opinion about the subsequent certificate.

This can naturally be applied to SPKI. As discussed in section 3.3.3 the verifier of an authorization needs a complete chain of certificates as proof of authorization. To express the trust in such a chain, each certificate $C_i$ in this chain has to include an opinion about the authenticity of its subject.

The only obstacle are names used as certificate subjects. Obviously one cannot assess the *key authenticity* of names and hence not calculate their trust opinion. However, in a valid certificate chain a name subject must eventually be bound to a key by a name cert. Hence the opinion of trust into a name is equivalent to the opinion about trust in all name certs needed to obtain a key for the name.

The verifier of an authorization can calculate the trust of the chain for use as parameter in its decision about a request to a protected resource, and for instance only accept request that provide chains whose trust belief exceed a certain threshold. Combined with SPKI's authorization delegation this provides a flexible way of delegating and verifying authorization that can model a wide range of organizational structures.

## 5. FAILURE AND SUCCESS OF SPKI

This paper has introduced SPKI and discussed the definition and semantics of its certificates, and revealed the flexibility, elegance and expressiveness of naming and authorization. It has furthermore analyzed how trust can be measured and assessed in certificate chain.

It has however not given an practical use of SPKI, simply because there is none. There has been research about using SPKI with various technologies and protocols, for instance DNS [17], HTTP [6] or sensor network [23], but to this days SPKI has not seen wide-spread adoption in real applications. Especially it has not replaced X.509 in application though that was an original intent of SPKI.

One may speculate about the reasons for this lack of deployment. Probably SPKI just came too late. By the time it was standardized and sufficiently researched, X.509 was already widely employed and had become the cryptographic backbone of the internet. With a cryptographic infrastructure at hand, there was little motivation to implement yet another one. Moreover SPKI did not offer a business model for companies, thus there was little interest to invest into the deployment of this standard.

A decade after the standardization of SPKI it seems unlikely that SPKI will ever happen to replace X.509 in important application. However, it may serve well in special domains and niche applications that have need of a simple and easy to implement public key standard. This paper may help developers of such applications to look beyond X.509 and possibly discard its complexity in favour of a really *simple public key infrastructure*.

## References

[1] M. Abadi. “On SDSI's Linked Local Name Spaces”. In: *CSFW*. Ed. by IEEE Computer Society. IEEE. 1997, pp. 98–108.

[2] A. Abdul-Rahman. “The PGP trust model”. In: *EDI-Forum: the Journal of Electronic Commerce* 10.3 (1997), pp. 27–31.

[3] H. Adkins. *An update on attempted man-in-the-middle attacks*. Google Inc. Aug. 29, 2011. Url: `http://googleonlinesecurity.blogspot.de/2011/08/update-on-attempted-man-in-middle.html`.

[4] U.S. Social Security Administration, ed. *Top names of the 2000s*. May 14, 2012. Url: `http://www.socialsecurity.gov/OACT/babynames/decades/names2000s.html`.

[5] J. Callas et al. *OpenPGP Message Format*. RFC 4880 (Standards Track). IETF, Nov. 2007. Url: `http://www.ietf.org/rfc/rfc4880.txt`.

[6] D. Clarke. “SPKI/SDSI HTTP Server / Certificate Chain Discovery in SPKI/SDSI”. Master's Thesis. Massachusetts Institute of Technology, 2001.

[7] D. Clarke et al. “Certificate Chain Discovery in SPKI/SDSI”. In: *Journal of Computer Security* 9.4 (2001), pp. 285–322.

[8] M. Copeland, J. Grahn, and D. Wheeler. *The GNU Privacy Handbook*. The Free Software Foundation. 1999. Url: `http://www.gnupg.org/gph/en/manual.html`.

[9] T. Dierks and E. Rescorla. *The Transport Layer SEcurity (TLS) Protocol Version 1.2*. RFC 4492 (Standards Track). IETF, Aug. 2008. Url: `http://www.ietf.org/rfc/rfc5246.txt`.

[10] W. Diffie and M. Hellman. “New Directions in Cryptography”. In: *IEEE Transactions on Information Theory* 6.22 (Nov. 1976), pp. 644–654.

[11] J. Elien. “Certificate Discovery Using SPKI/SDSI 2.0 Certificates”. Master's Thesis. Massachusetts Institute of Technology, 1998.

[12] C. Ellison. “Establishing Identity Without Certification Authorities”. In: *Proceedings of the 6th USENIX Security Symposium*. Ed. by USENIX. 1996, pp. 67–76.

[13] C. Ellison. *SPKI Requirements*. RFC 2692 (Experimental). IETF, Sept. 1999. Url: `http://www.ietf.org/rfc/rfc2692.txt`.

[14] C. Ellison et al. *SPKI Certificate Theory*. RFC 2693 (Experimental). IETF, Sept. 1999. Url: `http://www.ietf.org/rfc/rfc2693.txt`.

[15] Mozilla Foundation, ed. *Protection against fraudulent DigiNotar certificates*. MFSA 2011-34. Aug. 30, 2011. Url: `https://www.mozilla.org/security/announce/2011/mfsa2011-34.html`.

[16] P. Gutmann. *Everything you Never Wanted to Know about PKI but were Forced to Find Out*. 2002.

[17] T. Hasu and Y. Kortesniemi. “Implementing an SPKI Certificate Repository within the DNS”. In: *Poster Paper Collection of the Theory and Practice in Public Key Cryptography (PKC 2000* (2000), pp. 18–20.

[18] A. Jøsang. “An Algebra for Assessing Trust in Certification Chains”. In: *NDSS*. Ed. by The Internet Society. 1999.

[19] L. Kohnfelder. “Towards a Practical Public-key Cryptosystem”. Bachelor. Massachusetts Institute of Technology, May 1978.

[20] I. Lehti and P. Nikander. “Certifying Trust”. In: *Public Key Cryptography*. Ed. by Hideki Imai and Yuliang Zheng. Springer, 1998, pp. 83–98.

[21] J. Nightingale. *DigiNotar Removal Follow Up*. Mozilla Foundation. Sept. 2, 2011. Url: `http://blog.mozilla.org/security/2011/09/02/diginotar-removal-follow-up/`.

[22] Johnathan Nightingale. *Fraudulent *.google.com Certificate*. Mozilla Foundation. Sept. 6, 2011. Url: `http://blog.mozilla.org/security/2011/08/29/fraudulent-google-com-certificate/`.

[23] C. Pearce, V. Yin-Man Ma, and P. Bertok. “A secure communication protocol for ad-hoc wireless sensor network”. In: *ICISSNIP*. Ed. by IEEE Computer Society. IEEE. 2004, pp. 79–84.

[24] B. Ramsdell and S. Turner. *Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification*. RFC 5751 (Standards Track). IETF, Jan. 2010. Url: `http://www.ietf.org/rfc/rfc5751.txt`.

[25] R. Rivest. *S-Expressions*. Massachusetts Institute of Technology. May 4, 1997. Url: `http://people.csail.mit.edu/rivest/Sexp.txt`.

[26] R. Rivest. *SEXP---(S-expressions)*. Massachusetts Institute of Technology. May 4, 1997. Url: `http://people.csail.mit.edu/rivest/sexp.html`.

[27] R. Rivest and B. Lampson. *SDSI - A Simple Distributed Security Infrastructure*. Massachusetts Institute of Technology. Sept. 15, 1996. Url: `http://people.csail.mit.edu/rivest/sdsi10.html`.

[28] B. Schneier. *Beyond Fear. Thinking Sensibly About Security in an Uncertain World*. Springer Science+Business Media, 2006. ISBN: 978-0-387-02620-6.