TECHNION - Israel Institute of Technology
Computer Science Dept.

RELATIONS AMONG PUBLIC KEY
SIGNATURE SYSTEMS

by

Shimon Even* and Yacov Yacobi**

Technical Report #175
March 1980

* Computer Science Dept., Technion, Haifa, Israel
** Graduate Student, Electrical Engineering Dept., Technion, Haifa, Israel.
ABSTRACT

We attempt to establish the axioms and definitions of Public-Key Identification, Signature and Agreement Schemes (PKIS, PKSS and PKAS respectively, PKS is the collective name for all types).

The discussion is restricted to PKS in which there is no need for a third honest party (a 'judicator' [3]) to interfere in the usual operation of the system. The only role a judicator plays is in initiating a PKS and in settling disputes. The system does not require the release of secrets to the judicator.

Our definitions lead to the following main results:

(a) There is no PKAS.

(b) Every PKSS is a PKIS, but the converse is not necessarily true.

(c) Every PKSS which is based on a dialog between parties can be converted into a single transmission PKSS.
Discussion

We attempt to establish the axioms and definitions of Public-Key Identification, Signature and Agreement Schemes (PKIS, PKSS and PKAS respectively, PKS is the collective name for all types).

The discussion is restricted to PKS in which there is no need for a third honest party (a 'judicator' [3]) to interfere in the usual operation of the system. The only role a judicator plays is in initiating a PKS and in settling disputes. The system does not require the release of secrets to the judicator.

Throughout this paper we use the phrases 'hard to compute' and 'easy to compute' without explicit definitions, since our results are independent of the complexity measures used. Let us define first some new complexity notions to be used later in our discussion.

In every day practice we often encounter problems for which certain assumptions exist. These assumptions rule out some of the possible input instances, i.e. the required algorithm is guaranteed to receive as an input only a subclass of the input instances of the problem. It seems therefore natural to investigate the time complexity of the problem, provided the background assumptions hold.

Let us define this unconventional type of a problem [4], which we call 'promise problem'. (Some authors use the name 'birdy-problem', [5,6].)

In order to understand it better, first consider a conventional problem:

Input: x,

Property: P(x).

Where P is a predicate. A solution is an algorithm AL, which
halts with a 'yes' or 'no' answer such that:

$$\forall x[AL(x) = 'yes' \iff P(x)].$$

A promise problem has the following structure:

Input: $x$,
Promise: $Q(x)$,
Property: $P(x)$.

Where $P$ and $Q$ are predicates. Now, a solution is an algorithm $AL$ such that

$$\forall x[Q(x) \iff (AL(x) = 'yes' \iff P(x))].$$

Definition 1: Assume $B$ and $B_1$ are the following decision problems:

Problem $B$:
Data: $x_1, x_2$, integers,
Promise: $(\exists x)[Q(x, x_2)] \iff Q(x_1, x_2),$
where $Q$ is a predicate,
Property: $(\exists x)[Q(x, x_2)].$

Problem $B_1$:
Data: $x_2$, integer,
Property: $(\exists x)[Q(x, x_2)].$

$x_1$ is an essential datum of $B$ if $B$ is easily solvable, but $B_1$ is not.

Conclusion 1: $B \in P \implies B_1 \in NP.$

Conclusion 2: It is hard to find an essential datum, given the rest of the data. Analogously, we define essential datum for construction problems.

Definition 1': Assume $B$ and $B_1$ are the following construction problems:
Problem B:

Data: \(x_1, x_2\) integers,

Promise: \((\exists x, y)[Q(x, x_2, y)] \Rightarrow (\exists y)[Q(x_1, x_2, y)]\),

where \(Q\) is a predicate,

Task: Find \(y\) such that \((\exists x)[Q(x, x_2, y)]\), if such exists.

Problem \(B_1\):

Data: \(x_2\) integer.

Task: Find \(y\) such that \((\exists x)[Q(x, x_2, y)]\), if such exists.

\(x_1\) is an **essential datum** of \(B\) if \(B\) is easily solvable, but \(B_1\) is not.

Definition 2: A **One Way Function** (OWF) is an easily computable function \(f\) for which given a \(y\) it is hard to find an \(x\) such that \(f(x) = y\).

(See [1], for example.) We denote the fact that a function \(f\) is a OWF by an arrow above (i.e. \(\vec{f}\)).

For all types of PKS we assume the following:

**Hypothesis 1:**

(a) Every participant chooses some secret key \(z_i\) which only he knows.
(b) He uses a publically known OWF \(G\) to compute \(\vec{G}(z_i) = y_i\).
(c) He is conventionally identified by the judicator, and announces \(y_i\).
(d) If \(y_i\) is not yet used then the judicator accepts \(y_i\) as a means of identifying \(i\).
(e) From now on everybody knowing \(z_i\) is regarded as being \(i\), but the release of \(z_i\) should not be necessary for verifying \(i\)'s identity.

Let us define now the notion of identification in public key systems.

Definition 3: A **proof of identification** in public key systems is the outcome of a process of computations and transmissions between \(j\) and a party claiming to be \(i\), such that at the end of the process \(j\) knows
whether or not the other party holds \( z_i \). A process implementing this goal is called a Public Key Identification System (PKIS).

We now characterize the first type of a PKSS which implements what we call a Uni-Directional-Signature (UDS).

**Definition 4**: A UDS, by \( i \) of a message \( m \) which includes the time and date of the signature, is a word \( m_i \) which is an outcome of a process of calculations and transmissions, at the end of which it is known to the receiver \( j \). The process has the following properties:

(a) Given \( z_i \) and \( m \), there is OMF which yields \( m_i \). Also, given \( m, z_i \) is essential datum for the computation of a legitimate pair \((m, m_i)\). This must hold even if a polynomially bounded history of message-signature pairs is available, provided it does not contain \((m, m_i)\) explicitly. Also, it must hold even if a polynomially bounded set of keys \( \{z_j\} \) is known, but \( z_i \) is not in the set.

(b) The knowledge of \( y_i \) is sufficient for efficient verification of a given \((m, m_i)\) pair.

Note that in a UDS we do not need the restriction that it should be hard for \( i \), the signer, to fabricate some \( m' \) suitable for a given \( m_i \), because another pair \((m', m_i)\) does not contradict the fact that \((m, m_i)\) is sufficient to prove \( i \)'s obligation.

From Property (a) of Definition 5 it follows that a UDS can be implemented using one transmission only.

Let us define now the verification problem, VP, which implies directly that \( y_i \) is necessary for the verification process. Let \( Ω \) be the algorithm producing the signature.
Problem VP:

Data: \((m, m_1), y_1\),

Property: \(\exists z_1 [G(z_1) = y_1 \land \delta(z_1, m) = m_1]\).

The number \(i\) is not a part of VP's data. It is not relevant because \(z_1\) is a random variable independent of \(i\). Also, VP's property is not defined without \(y_1\), therefore \(y_1\) is necessary for the verification process. We conclude that a PKSS which implements a UDS has the following form. (\(\delta\) and \(E\) are the algorithms used by the signer and the verifier respectively.)

![Diagram](image)

**Figure 1:** A block diagram of a PKSS which implements a UDS.

We now turn our attention to another kind of PKSS which implements, what we call, a Bi-Directional-Signature (BDS).

**Definition 5:** A BDS of \(i\) before \(j\) on message \(m\), which includes time and date of signature, is a word \(m_{1j}\), which \(j\) (but perhaps not \(i\)) has as an outcome of a process of calculations and transmissions. The process has the following properties:
(a) There is an efficient computation of \( m_{ij} \) which uses \( z_i, z_j \) and \( m \). Also, each of \( z_i \) and \( z_j \) is essential datum for the computation of a legitimate pair \((m, m_{ij})\). This must hold even if a polynomially bounded history of message-signature pairs is available, provided it does not contain \((m, m_{ij})\) explicitly. Also, it must hold even if a polynomially bounded set of keys \( \{z_k\} \) is known, but \( z_i \) or \( z_j \) is not in the set.

(b) The knowledge of \( y_i \) and \( y_j \) is sufficient for efficient verification of a given \((m, m_{ij})\) pair.

From (a) one concludes that not only is it hard to forge a BDS on a given \( m \), but it is also hard for each one of \( i \) and \( j \) to pretend that they actually signed some other message \( m' \), or to fabricate some new pair. Also it assures that a BDS of \( i \) before \( j \) is untransferable to a BDS of \( i \) before \( k \), without \( i \)'s consent.

Like the case of UDS, it is easily shown, that each of \( y_i \) and \( y_j \) is necessary for the verification process of a BDS.

We conclude that PKSS which implements BDS has the following form:

![Diagram of PKSS which implements BDS](Figure 2: A PKSS which implements BDS (dotted lines denote data which may or may not be essential).)
Lemma 1: A PKSS which implements a BDS can be effectively converted into a PKSS which implements a UDS.

Proof: Given a PKSS implementing a BDS we build a PKSS which implements a UDS as follows: (denote the new parameters and operators by asterisks)

\[ z_1^* = (z_i, z_j); \quad G^*(z_i^*) = y_i^* = (y_i, y_j). \]

Finally, let \( D^* \) implement the dialog between \( D_1 \) and \( D_2 \) and output \( m_i^* = m_{ij} \).

Q.E.D.

Remarks:

(i) From Property (a) of Definition 5 one concludes that it must be hard to compute \((m, m_{ik})\) given \((m, m_{ij})\), \( z_j \) and \( z_k \). (i.e. the signature must be untransferable.) Also, the problem is easily avoided by including \( i, j \) and the time and date of signature in the message to be signed.

(ii) The construction of a BDS scheme is easy, assuming for example, that the RSA \([2]\) scheme is hard to crack. In this case two transmissions will do, where \( i \) starts, and the dotted lines of Fig. 2 can be omitted.

(iii) Every PKSS implementing a UDS or a BDS is also a PKIS (Definition 3). It is not clear whether the converse is true. For example, there exists a hypothetical possibility sketched in Appendix A, of a PKIS which is not a PKSS.

We now suggest what we believe to be a plausible definition of a PKAS. We then show that such a PKAS is not achievable, and therefore, conclude that BDS is the best substitute one can hope for.

Definition 6: A PKAS is a PKSS which implements a BDS so that it never happens that one of the parties can compute \( m_{ij} \), while the other cannot.

This point deserves elaboration. By Definition 5, \( m_{ij} \) is indeed a
mutual signature of an agreement. Definition 6 imposes additional symmetry on the process creating $m_{ij}$, which we believe is of great importance for contractual commercial relations.

Lemma 2: (If the judicator is not active during the ordinary operation of the system then) There is no PKAS.

Proof: Assume that, after $n$ communications, $i$ has sufficient information for efficient calculation of $m_{ij}$, but that this is not true for $n-1$ communications. We conclude that $j$ transmits the $n$-th communication, and therefore the first time $j$ has sufficient information is after $n'$ communications, where $n' \neq n$. This contradicts Definition 6.

Q.E.D.
APPENDIX A: A HYPOTHETICAL PKIS WHICH IS NOT A PKSS

Consider a system similar to the one sketched in Fig. 1, and assume an active eavesdropper having finite resources, such that for input of, say, length 100 bits it takes one minute to calculate $m_i$ given $m$ and $z_i$, and two minutes when $z_i$ is omitted.

In such a situation the system, clearly, cannot serve as a PKSS. However, it is still possible to use the system as a PKIS (Definition 3) as follows:

(a) The identifier, $j$, chooses at random some $m$, transmits it to the party which claims to be $i$, and requests that it calculates $m_i$ and transmit it to $j$ as soon as possible.

(b) Upon receiving the answer, $j$ checks the response time. If it is not more than one minute and if $E(m,m_i,y_i) = 'yes'$, then $j$ knows that the other party holds $z_i$, i.e. $j$ accepts that the other party is $i$. 
REFERENCES


