A Note on the Perfect Encryption Assumption in a Process Calculus

Roberto Zunino and Pierpaolo Degano

Dipartimento di Informatica, Università di Pisa, Italy
{zunino,degano}@di.unipi.it

Abstract. We consider a secrecy property in a simple process calculus with cryptographic primitives. The standard Dolev–Yao attacker is enhanced so that it can guess the key for decrypting an intercepted message. We borrow from the computational complexity approach to secrecy the assumptions that guessing succeeds with a given negligible probability and that the resources available to attackers are polynomially bound. Under these hypotheses we prove that the standard Dolev–Yao attacker is as powerful as the enhanced one.

1 Introduction

The analysis of security protocols has been the subject of a lot of recent work. While the problem has been approached in many different ways, the techniques used can be classified into two main classes. On the one side, we have results coming from *computational complexity theory* which offers a detailed, in–depth view of cryptosystems and protocols and deals with probability and algorithms. On the other side, *formal methods* provide abstractions that allow for mechanical proofs of protocol properties, but often require stronger assumptions (e.g. *perfect* or *unbreakable* encryption).

Some effort for bridging the gap has been started. Abadi and Rogaway [4] relate a formal equivalence between two terms to the computational infeasibility of distinguishing between their encodings into bit strings. Also, Troina et al. [13] follow a similar line and refine the equivalence of [4]. Backes and Jacobi's paper [5] is also related. These papers go from the computational towards the formal approach.

We instead proceed in the opposite direction: we use a process calculus to model cryptographic protocols and we explicitly allow the attacker to break encryptions by means of a *guessing* operation. Roughly speaking, once intercepted a message, the attacker can deduce the key to decrypt it with a given probability.

Computational reasoning predicts that guessing is hard, provided that the cryptosystem is good, the attacker has only a reasonably bounded computational power (e.g. the attacker is in $\mathcal{P}$–Time), and the keys are long enough. Many other factors affect the probability of guessing, but for the sake of simplicity, often one considers only those above. Also, the actual probability distribution is left implicit and the results about the robustness of protocols are proved taking these probabilities as a parameter. Here, we shall follow the same pattern.

I. Walukiewicz (Ed.): FOSSACS 2004, LNCS 2987, pp. 514–528, 2004.

We compare the traditional Dolev–Yao model (DY for short) in which guessing is not permitted against our enhancement ($\mathsf{DY}_{\mathbb{P}}$). The comparison is made easy by the fact that, if we forbid the use of the guessing operation, the $\mathsf{DY}_{\mathbb{P}}$ model collapses to DY. Moreover, since we only deal with discrete probability distributions, removing the guessing operation is equivalent to assuming the guessing probability to be null everywhere.

We give two definitions of *secrecy* for a protocol. Both are given in Sect. 4 and consider only the secrecy of a selected piece of data. One definition is the traditional one: it states that the attacker cannot learn a given secret by interacting with the protocol and by constructing/deconstructing the intercepted data with no bounds, except for encryptions being unbreakable. The other definition, instead, is taken from the computational complexity approach: the *probability* of learning a certain secret is a function of the key length, assuming the attacker is in $\mathcal{P}$–Time and breaking the cryptosystem is a problem not in $\mathcal{P}$–Time. Then, one studies the asymptotic behaviour of this function. Our first secrecy definition is tailored on the DY model, and our second one is apt for the $\mathsf{DY}_{\mathbb{P}}$ model.

Our main result shows that the two security definitions are equivalent. From the one hand, this result is not surprising: increasing the length of the keys and still keeping the attacker polynomially bounded (with respect to the key length) results in a virtually perfect encryption. From the other hand, under the standard hypothesis that guessing is hard, the traditional DY model has an accuracy comparable with that of the $\mathsf{DY}_{\mathbb{P}}$ model, in spite of being considerably simpler (see Sect. 4 for protocol evaluation in the $\mathsf{DY}_{\mathbb{P}}$ model).

In this note we make some assumptions in order to keep the presentation short and simple. In Sect. 5 we shall argue that many of them do not affect our main result. Also, the calculus we use is rather simple (its syntax and semantics are in Sect. 2 and 3), but can easily be extended to cover aspects not considered here.

2 A Simple Process Calculus: Syntax

We introduce a simple process calculus which can be used to model cryptographic protocols. This calculus is basically a variant of the π–calculus [10] enriched with cryptographic primitives, much alike the Spi–calculus [2]. Since we want to keep our calculus as simple as possible, we use only symmetric encryption in the calculus. Further extensions will be addressed in Sect. 5.

Let $\mathcal{N}$ be a denumerable set of *names* and let $\mathcal{V}$ be a denumerable set of *variables*. We use the letters n, m, o to range over $\mathcal{N}$ and the letters x, y, z to range over $\mathcal{V}$. Under these assumptions, we define *terms* in the following way. Terms represent the messages that are sent over the network when the protocol is run. Their meaning is standard.

$$
\begin{array}{lll}
L, M, N ::= 0 \mid 1 & & \text{bit} \\
\quad \mid x & & \text{variable} \\
\quad \mid n & & \text{name} \\
\quad \mid (M, N) & & \text{pair} \\
\quad \mid \{M\}_N & & \text{symmetric encryption}
\end{array}
$$

We now define *processes* in the following way:

$$
\begin{array}{rll}
P, Q, R ::= & \mathsf{nil} & \text{null process} \\
| & (x).P & \text{input} \\
| & \langle M \rangle.P & \text{output} \\
| & (P \mid Q) & \text{composition} \\
| & !\ P & \text{replication} \\
| & (\mathsf{new}\ n)P & \text{declaration} \\
| & \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q & \text{conditional} \\
| & \mathsf{split}\ M\ \mathsf{as}\ (x, y)\ \mathsf{in}\ P & \text{split} \\
| & \mathsf{decrypt}\ M\ \mathsf{as}\ \{x\}_N\ \mathsf{in}\ P & \text{symmetric decryption}
\end{array}
$$

The above syntax is quite standard, and readers familiar with other process calculi will find it rather straightforward. A simple informal description follows.

The nil process does not perform any operation. The input process $(x).P$ reads a value from the network, assigns it to the variable x and then behaves as P. The output process $\langle M \rangle.P$ sends a value over the network and then behaves as P. The parallel composition of two processes is denoted by $(P \mid Q)$. The replication $!\ P$ behaves as the parallel composition of an unlimited number of P processes. We write $(\mathsf{new}\ n)P$ for the creation of new names (i.e., new keys, fresh nonces, etc.). The conditional tests whether two terms are equal. Split and decryption are used to destruct pair and encryption terms. Note that the decryption requires the (secret) key to succeed.

The most important features of this calculus are the following:

- All the input and output operation use the same *global* public channel, unlike some other calculi, as the π–calculus or the Spi–calculus, where input and output occur on given channels.
  While this makes impossible, for instance, to specify the destination for a message (which is very inconvenient from a programmer's point of view), it models the standard Dolev–Yao assumption which states that the attacker is able to reroute messages.
- There are no private channels in our calculus. Private channels could be used to model *secure links* and will be discussed in Sect. 5.

A variable is said to be *bound* if it occurs under an input prefix, split, or decryption; otherwise it is said to be *free*. Similarly, a name is *bound* if it occurs under a declaration $(\mathsf{new}\ n)$, otherwise it is *free*. We write $\mathsf{fv}(P), \mathsf{bv}(P), \mathsf{fn}(P), \mathsf{bn}(P)$ respectively for (the set of) the free variables, the bound variables, the free names, and the bound names of the process P.

3 A Simple Process Calculus: Semantics

We define an *attacker-aware* semantics of our process calculus. This means that our semantics explicitly assumes the presence of an attacker and models the interaction between it and the system which executes a certain protocol (i.e. a process P).

We first make two assumptions about what the attacker can or can not do:

- As the Dolev–Yao attacker, ours can intercept and learn messages as they are sent over the network. The attacker can then send over the network terms built from the known messages by using the following *operations*: pairing, splitting of pairs, encryption, and decryption. This also implies the attacker is able to reroute, discard, and replace messages, possibly pretending they are from someone else;
- Unlike the Dolev–Yao attacker, ours can also *guess* a key which has been used to encrypt some message. This special operation, however, only succeeds with a given probability.

3.1 The Attacker

We introduce the following *entailment rules*, that define the operations of attacker. Each rule can be read as: "the attacker can build the term of the right hand side of the arrow provided it knows the terms on the left hand side". (We comment below on only having names as keys.)

$$\begin{array}{ll} \mathsf{DYCons} & M, N \overset{1}{\longmapsto}_{\mathsf{DY}} (M, N) \\ \mathsf{DYFst} & (M, N) \overset{1}{\longmapsto}_{\mathsf{DY}} M \\ \mathsf{DYSnd} & (M, N) \overset{1}{\longmapsto}_{\mathsf{DY}} N \end{array} \qquad \begin{array}{ll} \mathsf{DYEnc} & M, n \overset{1}{\longmapsto}_{\mathsf{DY}} \{M\}_n \\ \mathsf{DYDec} & \{M\}_n, n \overset{1}{\longmapsto}_{\mathsf{DY}} M \end{array}$$

$$\mathsf{DYGuess}\ \{M\}_n \overset{p}{\longmapsto}_{\mathsf{DY}} n$$

Each arrow has a label, expressing the probability of the operation to succeed. The first five rules are the traditional Dolev–Yao rules and get probability 1 as they never fail. Instead, rule DYGuess allows the attacker to guess the secret key n with probability p.

However, a question arises: what should we choose as *success probability* p for DYGuess?

In the computational complexity approach, the guessing probability depends on many factors, such as which cryptosystem was used for the encryption, the length of the key used, whether the attacker knows other messages encrypted with the same key, whether the attacker knows the plaintext, the amount of computational resources (i.e. time) spent by the attacker, and so on. Even if all these factors are taken into account, there is no handy formula for the guessing probability. Therefore, in order to keep our system simple, we make some further assumptions:

- We constrain encryptions to use only names as keys: all the encryptions thus must be on the form $\{M\}_n$. This can be enforced by a simple *type system*. Furthermore, we assume that names are encoded as bit strings of the same length η[1].

[1] In the computational complexity approach, η is typically referred to as the *security parameter*. Here, we simply identify it with the key length.

– We assume that the guessing probability depends only on η: we write it as $\mathbb{P}_{\mathsf{guess}}(\eta)$. In particular, it does not depend on the result (success or failure) of previous DYGuess operations. Most importantly, here we stipulate that the attacker consumes one unit of computational resources in order to perform the guess. We shall further discuss this point in Sect. 5.

Intuitively, the function $\mathbb{P}_{\mathsf{guess}}$ describes the strength of the cryptosystem: if $\mathbb{P}_{\mathsf{guess}}(\eta)$ is small, the cryptosystem is strong. Typically, in the computational complexity approach, a cryptosystem is regarded as *safe* if the function $\mathbb{P}_{\mathsf{guess}}$ (or other similar function) "quickly" approaches zero as soon as η increases. In other words, this means that the strength of the cryptosystem rapidly increases as soon as larger keys are used. We shall use this kind of asymptotic assumption on $\mathbb{P}_{\mathsf{guess}}$ in Sect. 4.

We do not put further assumptions on $\mathbb{P}_{\mathsf{guess}}(\eta)$. Instead, we shall consider it a *free parameter*.

3.2 The Transition System

In order to define the semantics of our calculus, we use a *labeled transition system* (LTS). Its states are composed of:

– the attacker's knowledge $\mathcal{K}$, which represents the set of terms that the attacker has learnt or built so far;
– a process P, which represents the current state of the execution of the protocol;
– an optional pair (p, N), used to model the fact that the attacker is about to learn the term N with probability p.

We write $(\mathcal{K}, P)$ (or, if the optional part is present, $(\mathcal{K}, P)^{p,N}$) to represent a generic state of the LTS.

We now give the intuition underlying our transitions. A first kind of transition models a "standard" action of a process P:

$$(\mathcal{K}, P) \xrightarrow{\mu} (\mathcal{K}', P')$$

Initially, the attacker has knowledge $\mathcal{K}$. Then, P performs an action and becomes P'. If the action is the output of a message M, the new knowledge is $\mathcal{K}' = \mathcal{K} \cup \{M\}$, otherwise $\mathcal{K}' = \mathcal{K}$.

Another kind of transition models an action of the attacker:

$$(\mathcal{K}, P) \xrightarrow{\spadesuit} (\mathcal{K}, P)^{p,N}$$

This transition represents the choice of some operation *op* the attacker is going to apply to the terms it knows. The superscript p is the probability to get the result N ($p \neq 1$ only when the operation is a *guess*). From the target state of a $\xrightarrow{\spadesuit}$ transition, two transitions exit. One transition represents the success of operation *op* and is on the form

$$(\mathcal{K}, P)^{p,N} \xrightarrow[p]{\spadesuit_{\mathsf{s}}} (\mathcal{K} \cup \{N\}, P)$$

The other transition instead represents the failure of *op*:

$$(\mathcal{K}, P)^{p,N} \xrightarrow[1-p]{\spadesuit_{\mathsf{f}}} (\mathcal{K}, P)$$

Both these transitions are labeled with the corresponding probability.

We now define the transitions $\xrightarrow{\mu}$, $\xrightarrow{\spadesuit}$ and $\xrightarrow{\spadesuit_{\mathsf{s/f}}}$.

Process Actions ($\xrightarrow{\mu}$). It is convenient to partition the set of names $\mathcal{N}$ into equivalence classes. We assume that α–conversion of a name is only performed within its equivalence class. We write $n \sim_\alpha m$ when n and m belong to the same equivalence class. We also extend homomorphically the relation $\sim_\alpha$ to terms.

We write $P\{M/x\}$ for the substitution of the term M at every free occurrence of x in P, possibly α–converting bound names if necessary. We also write $M \sqsubseteq \mathcal{K}$ for $\exists N.\ M \sim_\alpha N \in \mathcal{K}$.

In order to keep our system simple, we consider processes up to the *structural congruence* relation $\equiv$, defined as the minimum congruence on the set of processes $\mathcal{P}$ including (our constrained) α-conversion, and such that $(\mathcal{P}/_{\equiv}, |, \mathsf{nil})$ is an abelian monoid.

We now give the rules for $\xrightarrow{\mu}$, representing one action performed by P. The transition label μ can be either a name or a special symbol τ.

$$\mu\mathsf{In}\ \frac{M \in \mathcal{K}}{(\mathcal{K}, (x).P) \xrightarrow{\tau} (\mathcal{K}, P\{M/x\})} \qquad \mu\mathsf{Out}\ \frac{}{(\mathcal{K}, \langle M \rangle.P) \xrightarrow{\tau} (\mathcal{K} \cup \{M\}, P)}$$

$$\mu\mathsf{Decl}\ \frac{n \text{ does not occur in any term of } \mathcal{K}}{(\mathcal{K}, (\mathsf{new}\ n)P) \xrightarrow{n} (\mathcal{K}, P)} \qquad \mu\mathsf{Rep}\ \frac{}{(\mathcal{K}, !\ P) \xrightarrow{\tau} (\mathcal{K}, P \mid !\ P)}$$

$$\mu\mathsf{Par}\ \frac{(\mathcal{K}, P) \xrightarrow{\mu} (\mathcal{K}', P') \quad \mu = \tau \vee \mu \notin \mathsf{fn}(Q)}{(\mathcal{K}, P|Q) \xrightarrow{\mu} (\mathcal{K}', P'|Q)}$$

$$\mu\mathsf{Then}\ \frac{}{(\mathcal{K}, \mathsf{if}\ M = M\ \mathsf{then}\ P\ \mathsf{else}\ Q) \xrightarrow{\tau} (\mathcal{K}, P)}$$

$$\mu\mathsf{Else}\ \frac{M \neq N}{(\mathcal{K}, \mathsf{if}\ M = N\ \mathsf{then}\ P\ \mathsf{else}\ Q) \xrightarrow{\tau} (\mathcal{K}, Q)}$$

$$\mu\mathsf{Dec}\ \frac{}{(\mathcal{K}, \mathsf{decrypt}\ \{M\}_n\ \mathsf{as}\ \{x\}_n\ \mathsf{in}\ P) \xrightarrow{\tau} (\mathcal{K}, P\{M/x\})}$$

$$\mu\mathsf{Split}\ \frac{}{(\mathcal{K}, \mathsf{split}\ (M, N)\ \mathsf{as}\ (x, y)\ \mathsf{in}\ P) \xrightarrow{\tau} (\mathcal{K}, P\{M/x\}\{N/y\})}$$

These rules are rather straightforward. We only note that the $\mu\mathsf{In}$ rule requires $M \in \mathcal{K}$, thus modeling the attacker sending a known term to the process. Rule $\mu\mathsf{Out}$ instead makes the attacker learn the term M. Rule $\mu\mathsf{Decl}$ and $\mu\mathsf{Par}$ ensure fresh names are generated each time a $(\mathsf{new}\ n)$ is executed.

Moreover, we note that if $(\mathcal{K}, P)$ is closed (i.e., P has no free variables and $\mathcal{K}$ only contains ground terms) all the states reachable from it are also closed. Also note that in the rules above the terms M and N are ground, provided that the source state is closed.

Attacker Actions. We can now define the $\xrightarrow{\spadesuit}$ and $\xrightarrow{\spadesuit_{s/f}}$ transitions as follows.

$$\spadesuit\mathsf{Decision}_1 \ \frac{M \in \mathcal{K} \quad M \overset{p}{\longmapsto}_{\mathsf{DY}} N}{(\mathcal{K}, P) \xrightarrow{\spadesuit} (\mathcal{K}, P)^{p,N}} \qquad \spadesuit\mathsf{Decision}_2 \ \frac{L, M \in \mathcal{K} \quad L, M \overset{p}{\longmapsto}_{\mathsf{DY}} N}{(\mathcal{K}, P) \xrightarrow{\spadesuit} (\mathcal{K}, P)^{p,N}}$$

$$\spadesuit\mathsf{Success} \ \frac{}{(\mathcal{K}, P)^{p,N} \xrightarrow[p]{\spadesuit_s} (\mathcal{K} \cup \{N\}, P)} \qquad \spadesuit\mathsf{Failure} \ \frac{}{(\mathcal{K}, P)^{p,N} \xrightarrow[1-p]{\spadesuit_f} (\mathcal{K}, P)}$$

The $\xrightarrow{\spadesuit}$ transitions model the choice of 1) which entailment rule the attacker is going to apply, and 2) which arguments it is applied to. Once the choice is performed, the $\spadesuit$Success and $\spadesuit$Failure rules model the actual application of the entailment rule. In case of success, the term N is added to the attacker's knowledge. Note that the transition $\xrightarrow[p]{\spadesuit_s}$ ($\xrightarrow[1-p]{\spadesuit_f}$) records the success (failure) probability, inherited by the entailment rule applied.

4 Protocol Evaluation

Having established our attacker–aware semantics, we can evaluate the strength of a given protocol. We actually focus on checking whether a protocol guarantees *secrecy*. This property can be easily expressed in our model because the states of our LTS explicitly expose the knowledge of the attacker $\mathcal{K}$. This allows us to check whether the attacker knows a term M by simply checking whether $M \sqsubseteq \mathcal{K}$.

We define the probability that the attacker, having initial knowledge $\mathcal{K}$, will learn a given term M by interacting with a process P for a *limited* period. In order to accomplish this, we put a bound t on the number of transitions. This bound limits the sum of 1) the number of interactions between the attacker and the process P (i.e., of $\xrightarrow{\mu}$ transitions) and 2) the number of operations the attacker can perform (i.e., of $\xrightarrow{\spadesuit}\xrightarrow{\spadesuit_{s/f}}$ pairs of transitions).

We denote that probability by $\mathbb{P}^{\,t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K}, P)$. For brevity, we often omit $\mathbb{P}_{\mathsf{guess}}(\eta)$, insisting that it is a free parameter.

In order to define $\mathbb{P}^{\,t}_{M}(\mathcal{K}, P)$, we make the following assumptions:

- $(\mathcal{K}, P)$ is closed and $\mathcal{K}$ is nonempty;
- $\mathsf{fn}(P) \cap \mathsf{bn}(P) = \emptyset$; also, names occurring in declarations belong to distinct equivalence classes. In this way, we can track the origin of names. For instance, we can check whether $P = !\ (\mathsf{new}\ n).\langle n \rangle.\mathsf{nil} \mid Q$ discloses an instance of n by checking whether $n \sqsubseteq \mathcal{K}'$ for some state $(\mathcal{K}', P')$ reachable from $(\mathcal{K}, P)$ where n does not occur in $\mathcal{K}$;
- among the transitions outgoing from a state and labeled by μ or $\spadesuit$, we consider only the one leading to the best state for the attacker. This is fairly standard in formal models: a protocol is regarded as *unsafe* if some execution trace leads to the disclosure of a secret. An alternative probabilistic approach will be discussed in Sect. 5.

Definition 1. $\mathbb{P}^{\,t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K},P)$ *is defined by induction on* t *by the following equations*[2].

$$\mathbb{P}^{\,t}_{M}(\mathcal{K},P)=1 \qquad \text{if } M\sqsubseteq\mathcal{K} \tag{1}$$

$$\mathbb{P}^{\,0}_{M}(\mathcal{K},P)=0 \tag{2}$$

$$\mathbb{P}^{\,t}_{M}(\mathcal{K},P)=\max\Big(\Big\{\mathbb{P}^{\,t-1}_{M}(\mathcal{K}',P') \;\Big|\; (\mathcal{K},P)\xrightarrow{\mu}(\mathcal{K}',P')\Big\}\cup \Big\{\mathbb{P}^{\,t}_{M}(\mathcal{K},P)^{p,N} \;\Big|\; (\mathcal{K},P)\xrightarrow{\spadesuit}(\mathcal{K},P)^{p,N}\Big\}\Big) \tag{3}$$

$$\mathbb{P}^{\,t}_{M}(\mathcal{K},P)^{p,N}=\sum\Big\{\Big|q*\mathbb{P}^{\,t-1}_{M}(\mathcal{K}',P) \;\Big|\; (\mathcal{K},P)^{p,N}\xrightarrow[q]{\spadesuit_{\mathsf{s/f}}}(\mathcal{K}',P)\Big|\Big\} \tag{4}$$

Equation (1) checks whether M was disclosed. Equation (2) instead checks whether M was not disclosed and time ran out (i.e., $t=0$). Equation (3) chooses the transition which is the best for the attacker; the chosen transition may be a process action $\xrightarrow{\mu}$ or an attacker action $\xrightarrow{\spadesuit}$ corresponding to some operation *op*. Equation (4) considers both the cases when the operation *op* succeeds $\Big((\mathcal{K},P)^{p,N}\xrightarrow[p]{\spadesuit_{\mathsf{s}}}(\mathcal{K}',P)\Big)$ and fails $\Big((\mathcal{K},P)^{p,N}\xrightarrow[1-p]{\spadesuit_{\mathsf{f}}}(\mathcal{K},P)\Big)$. Then, the probability of discovering M from $(\mathcal{K},P)^{p,N}$ is the sum of the probability of discovering M from $(\mathcal{K}',P)$ weighted by p, and from $(\mathcal{K},P)$, weighted by $1-p$. Equation (4) could also be written as

$$\mathbb{P}^{\,t}_{M}(\mathcal{K},P)^{p,N}=p\cdot\mathbb{P}^{\,t-1}_{M}(\mathcal{K}\cup\{N\},P)+(1-p)\cdot\mathbb{P}^{\,t-1}_{M}(\mathcal{K},P)$$

The right hand sides of the equations in Definition 1 do not depend on the choice of names, i.e. they are not affected by our constrained version of α–conversion (note the use of the relation $\sqsubseteq$ in (1)). Also, from any state $(\mathcal{K},P)$, only finitely many transitions $\xrightarrow{\mu}$ and $\xrightarrow{\spadesuit}$ exit (up to α–conversion). Therefore, in (3), max is applied to a finite set. Moreover that set is nonempty because there is always at least the $\xrightarrow{\spadesuit}$ transition corresponding to the DYCons of some term belonging to the nonempty knowledge $\mathcal{K}$. Thus $\mathbb{P}^{\,t}_{M}(\mathcal{K},P)$ is well–defined.

A Probabilistic Secrecy Notion. The probability $\mathbb{P}^{\,t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K},P)$ measures the strength of a protocol as a function of the parameters $t,\mathcal{K}$ and $\mathbb{P}_{\mathsf{guess}}(\eta)$. We now aim for a definition of *safe* protocol which abstracts from those parameters.

In the computational complexity approach, a protocol is regarded as *safe* if, for any attacker with polynomially bounded resources (with respect to the security parameter η), the probability of disclosing a secret (as a function of η) is *negligible*, provided that the cryptosystem is strong, i.e. it can be broken only with a *negligible* probability by such an attacker. A function is said *negligible* if it approaches zero faster than any rational function. Formally:

Definition 2. *We say that a function* $f:\mathbb{N}\to\mathbb{R}$ *is negligible iff*

$$\forall k\in\mathbb{N}\;\exists\eta_0\in\mathbb{N}\;\forall\eta\in\mathbb{N}.\;\eta>\eta_0\implies|f(\eta)|<\eta^{-k}$$

[2] We assume that, whenever two equations overlap, the topmost one applies.

For example, $2^{-\eta}$ is negligible.

We map the definition of *safe* protocol surveyed above into our model. In order to (polynomially) bound the resources of the attacker, we let $t = \eta^k$. To us, a strong cryptosystem is such that $\mathbb{P}_{\mathsf{guess}}$ is negligible. Finally, we assume that the attacker initially knows only the terms 0 and 1. We can now define *safety* in the following way:

Definition 3. *We say that a protocol P is $\mathsf{DY}_{\mathbb{P}}$–safe with respect to a secret term M iff, for any $\mathbb{P}_{\mathsf{guess}} : \mathbb{N} \to [0,1]$,*

$$\mathbb{P}_{\mathsf{guess}} \text{ negligible} \implies \forall k \in \mathbb{N}.\ \mathbb{P}^{\eta^k}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\{0,1\}, P) \text{ negligible}$$

The above definition can be read as: P is safe if, provided we are using a strong cryptosystem, any attacker with polynomially bounded resources has only negligible probability of discovering M. Note that, if $\mathbb{P}_{\mathsf{guess}}(\eta) > 0$, bounding the resources of the attacker ($t = \eta^k$) is crucial, because for any protocol P which outputs a term where M occurs, we have

$$\lim_{t\to\infty} \mathbb{P}^{t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K}, P) = 1\ .$$

Indeed, an unbounded attacker can repeatedly apply the DYGuess rule on any encryption until it eventually succeeds.

Towards a Comparison against Traditional Secrecy. We compare the above definition of $\mathsf{DY}_{\mathbb{P}}$–*safe* process with the standard notion of secrecy used in formal models ($\mathsf{DY}_{\mathsf{std}}$–*safety*), which assumes a deterministic Dolev–Yao attacker.

Definition 4. *We say that a protocol P is* $\mathsf{DY}_{\mathsf{std}}$–safe *w.r.t. a secret M iff*

$$\forall \mathcal{K}, P'.\ \ (\{0,1\}, P) \longrightarrow^{*} (\mathcal{K}, P') \implies M \not\in \mathcal{K}$$

where the arrow $\longrightarrow$ stands for either a $\xrightarrow{\mu}$ transition or a pair $\xrightarrow{\spadesuit}\xrightarrow{\spadesuit_{\mathsf{s}}}$ derived from a deterministic *Dolev–Yao rule (i.e. not using a* DYGuess*).*

Note that this definition does *not* bound the number of transitions and therefore, unlike definition 3, is not resource–conscious.

It is convenient to rephrase the $\mathsf{DY}_{\mathsf{std}}$–*safety* property using $\mathbb{P}^{t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K}, P)$ in order to simplify the comparison between Definitions 3 and 4. A trivial way of making the DYGuess rule harmless is assuming a null guessing probability: this makes our model deterministic.

Definition 5. *Let $\mathbb{P}_0$ be the constant null function. We say that a protocol P is* DY*–safe with respect to a secret M iff*

$$\forall t, \eta \in \mathbb{N}.\ \mathbb{P}^{t}_{M,\mathbb{P}_0(\eta)}(\{0,1\}, P) = 0\ .$$

Proposition 1. *The probability* $\mathbb{P}^{\,t}_{M,\mathbb{P}_0(\eta)}(\mathcal{K}, P)$ *is either* 0 *or* 1.

Proof. Trivial induction. □

Moreover, it it easy to show by induction that definitions 4 and 5 are indeed equivalent, as the following proposition states:

Proposition 2. *P is $\mathsf{DY}_{\mathsf{std}}$–safe (w.r.t. M) iff P is DY–safe (w.r.t. M).*

Proposition 2 allows us to compare $\mathsf{DY}_{\mathbb{P}}$–safety against $\mathsf{DY}_{\mathsf{std}}$–safety by instead comparing $\mathsf{DY}_{\mathbb{P}}$–safety against DY–safety as shown in the next section.

4.1 Comparing DY and $\mathsf{DY}_{\mathbb{P}}$ Attackers

The following obvious lemma says that the probability of discovering a secret increases with the power of the attacker; its proof is by easy induction on t.

Lemma 1 (Monotonicity). *If,* $\forall \eta \in \mathbb{N}.\ \mathbb{P}_{\mathsf{g1}}(\eta) \leq \mathbb{P}_{\mathsf{g2}}(\eta)$ *and* $t_1 \leq t_2$*, then*

$$\forall \eta \in \mathbb{N}.\ \mathbb{P}^{\,t_1}_{M,\mathbb{P}_{\mathsf{g1}}(\eta)}(\mathcal{K}, P) \leq \mathbb{P}^{\,t_2}_{M,\mathbb{P}_{\mathsf{g2}}(\eta)}(\mathcal{K}, P)$$

Our main result states the equivalence of $\mathsf{DY}_{\mathbb{P}}$–safety and DY–safety. We simply sketch its proof.

Theorem 1. *P is $\mathsf{DY}_{\mathbb{P}}$–safe (w.r.t. M) if and only if P is DY–safe (w.r.t. M).*

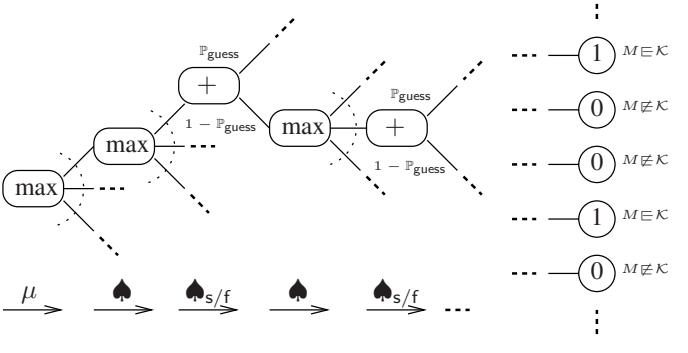
Proof (sketch). We rewrite the statement in the following way:

$$P \text{ is } \underline{\text{not}}\ \mathsf{DY}_{\mathbb{P}}\text{–safe w.r.t. } M \iff P \text{ is } \underline{\text{not}}\ \mathsf{DY}\text{–safe w.r.t. } M$$

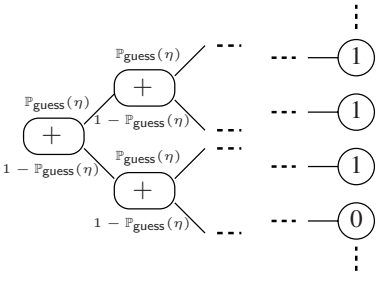
($\Leftarrow$) We have that, for some $\varepsilon > 0$, $\exists t \in \mathbb{N}.\ \mathbb{P}^{\,t}_{M,\mathbb{P}_0(\eta)}(\{0,1\}, P) = \varepsilon$. Therefore we can apply the monotonicity lemma and state that, for every $\eta > t$, $\mathbb{P}^{\,\eta}_{M,\mathbb{P}_0(\eta)}(\{0,1\}, P) \geq \varepsilon$. This implies that $\mathbb{P}^{\,\eta}_{M,\mathbb{P}_0(\eta)}(\{0,1\}, P)$ is not negligible (as a function of η). Since $\mathbb{P}_0$ is negligible, we conclude that P is not $\mathsf{DY}_{\mathbb{P}}$–safe.

($\Rightarrow$) By hypothesis, for some k and some negligible $\mathbb{P}_{\mathsf{guess}}$, $\mathbb{P}^{\,\eta^k}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\{0,1\}, P)$ is not negligible.

We now examine the definition of $\mathbb{P}^{\,t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K}, P)$. Given $t, M, \mathbb{P}_{\mathsf{guess}}(\eta), \mathcal{K}$, and P, we can fully expand the definition to form a tree like the following one:



We now simplify this tree by removing every max node from it and simply replacing it with one of its children which evaluate to the maximum. This new tree is made only of weighted sum nodes and leaf nodes. Simplify further the tree as follows: remove all the sums with weights 0 and 1 derived from Dolev–Yao rules other than the DYGuess rule; replace them with the success subtree (the one with weight 1). The resulting tree is thus formed only by weighted sums with weights $\mathbb{P}_{\mathsf{guess}}(\eta)$ and $1 - \mathbb{P}_{\mathsf{guess}}(\eta)$ and leaves.



Now we consider how the the simplified tree corresponding to $\mathbb{P}^{\eta^k}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\{0,1\}, P)$ changes as η increases. For every η, we write T^η and Π^η respectively for the simplified tree and the set of its paths. For every $\pi \in \Pi^\eta$ we define $\mathsf{weight}(\pi)$ be the product of all the labels of the edges of the path; we also define $\mathsf{length}(\pi)$ as the length of the path and $\mathsf{result}(\pi)$ as the value of the leaf at the end of the path. It turns out that $\mathbb{P}^{t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K}, P) = \sum\{\mathsf{weight}(\pi) \mid \pi \in \Pi \wedge \mathsf{result}(\pi) = 1\}$. Also, we write π^η_{fail} for the path of Π^η that represents the event in which the DYGuess rule always fails: π^η_{fail} has weight $\mathsf{weight}(\pi^\eta_{\mathsf{fail}}) = (1 - \mathbb{P}_{\mathsf{guess}}(\eta))^{\mathsf{length}(\pi^\eta_{\mathsf{fail}})}$.

We now claim that, for some η, $\mathsf{result}(\pi^\eta_{\mathsf{fail}})$ is 1. The proof is by contradiction. Assume $\mathsf{result}(\pi^\eta_{\mathsf{fail}}) = 0$ for every η: we obtain

$$\mathbb{P}^{\eta^k}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\{0,1\}, P) = \sum\{\mathsf{weight}(\pi) \mid \pi \in \Pi \wedge \mathsf{result}(\pi) = 1\} =$$
$$\sum\{\mathsf{weight}(\pi) \mid \pi \in \Pi \wedge \pi \neq \pi^\eta_{\mathsf{fail}} \wedge \mathsf{result}(\pi) = 1\} \leq$$
$$\sum\{\mathsf{weight}(\pi) \mid \pi \in \Pi \wedge \pi \neq \pi^\eta_{\mathsf{fail}}\} = \; 1 - \mathsf{weight}(\pi^\eta_{\mathsf{fail}}) =$$
$$1 - \Big(1 - \mathbb{P}_{\mathsf{guess}}(\eta)\Big)^{\mathsf{length}(\pi^\eta_{\mathsf{fail}})} \leq \; 1 - \Big(1 - \mathbb{P}_{\mathsf{guess}}(\eta)\Big)^{2\eta^k}$$

which can be shown to be negligible thus reaching a contradiction.

By establishing $\mathsf{result}(\pi^\eta_{\mathsf{fail}}) = 1$ for some η, we actually found an attack to the protocol which does not use the DYGuess rule. This attack can be used to show that P is not DY–safe. It is sufficient to lift the path π^η_{fail} back to the unsimplified tree and observe all the choices for the max nodes. Then, we can expand the definition of $\mathbb{P}^{t}_{M,\mathbb{P}_0(\eta)}(\{0,1\}, P)$ with $t = \eta$ that yields to an isomorphic tree. Under the same choices for the max nodes made before, we have $\mathbb{P}^{\eta}_{M,\mathbb{P}_0(\eta)}(\{0,1\}, P) = 1$. □

Theorem 1 and Proposition 2 state that the definitions of $\mathsf{DY}_{\mathsf{std}}$–safety, DY–safety and $\mathsf{DY}_{\mathbb{P}}$–safety are equivalent. Intuitively, an attacker with unbounded

resources, but no hope of guessing a key, is as powerful as an attacker that might guess a key (with a small probability) but only for a bounded amount of time. This equivalence, however, only holds asymptotically (with respect to the key length). As a matter of fact, the ability formal methods have to always closing the attacker knowledge (under the standard Dolev–Yao operations) subsumes the hardly mechanizable reasoning made in the computational approach.

5 Extensions and Future Work

Our model can easily be extended to include other attacker actions and cryptographic primitives. We briefly consider some of them.

Private Channels. It is trivial to allow private channels to model secure links by borrowing from the π or the Spi–calculus. Terms sent through the private channels do not affect the knowledge of the attacker $\mathcal{K}$, therefore modifying our attacker–aware semantics is easy.

Breaking without Guessing. In our model we allowed the attacker to *guess*, i.e. to deduce the key from a given encryption. We could also allow the attacker to *break* encryptions and get the plaintext without guessing the key. We can do this by adding the rule

$$\mathsf{DYBreak} \quad \{M\}_n \overset{p}{\longmapsto}_{\mathsf{DY}} M$$

As for *guessing*, we should assume p to be some negligible function $\mathbb{P}_{\mathsf{break}}(\eta)$.

Blind Guessing. We could also allow *blind guessing*, i.e. guessing without using any previous knowledge.

$$\mathsf{DYBlindGuess} \quad \overset{p}{\longmapsto}_{\mathsf{DY}} n$$

This rule differs from DYGuess in that it allows the attacker to guess, e.g., a key that has not yet been used by the protocol, or a nonce. Therefore, it is strictly more powerful than the DYGuess rule. As for *breaking*, p should be negligible.

Other Cryptographic Primitives. So far, we only considered symmetric encryptions. We could also model asymmetric encryption and digital signatures by adding, beyond the standard Dolev–Yao rules for asymmetric cryptography, other rules that explicitly break the cryptosystem. Many different rules can be used: we just give an example. We write n^+ and n^- for public and private keys, respectively.

$$\mathsf{DYInvert} \quad n^+ \overset{p}{\longmapsto}_{\mathsf{DY}} n^- \qquad \mathsf{DYFakeSign} \quad M, n^+ \overset{q}{\longmapsto}_{\mathsf{DY}} [M]_{n^-}$$

As above, p and q should be negligible.

Non Constant–Cost Operations. In the definition of $\mathbb{P}^{\,t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta)}(\mathcal{K}, P)$ the number t is decreased by one unit for each application of an entailment rule. So, every operation of the attacker has a constant cost.

This is an oversimplification; a more adequate modeling, in particular for guessing, can be obtained as follows. A simple manner is to assume that the operations of the attacker require a non unitary amount of resources; making them possibly depend on η. Another way is to relate the amount of resources consumed, e.g. for guessing, to the probability of success of the operation involved. For instance, we could have the following DYGuess rule.

$$\mathsf{DYGuess} \quad \{M\}_n \overset{\mathbb{P}_{\mathsf{guess}}(\eta,r)}{\underset{r}{\longmapsto}_{\mathsf{DY}}} n$$

The quantity r represents the amount of resources the attacker may decide to spend in guessing. Depending on the parameter r, a value is computed that expresses the probability of success. Thus, $\mathbb{P}_{\mathsf{guess}}(\eta, r)$ now takes into account both the length of the keys and the time spent. The more resources are consumed, the more likely guessing is; thus we require $\mathbb{P}_{\mathsf{guess}}(\eta, r)$ to be monotonic on r. We change the definition of $\mathbb{P}^{\,t}_{M,\mathbb{P}_{\mathsf{guess}}(\eta,-)}(\mathcal{K}, P)$ so that, whenever an entailment rule requiring r resources is applied, the number t is decreased by r units (obviously, $1 \leq r \leq t$, thus only finitely branching fragments of the transition system are considered). Of course, we have strong cryptosystems, i.e., for any $c \in \mathbb{N}$, the probability $\mathbb{P}_{\mathsf{guess}}(\eta, \eta^c)$ is negligible as a function of η. Accordingly, the definition of $\mathsf{DY}_{\mathbb{P}}$–safety requires that for any such $\mathbb{P}_{\mathsf{guess}}$ and for any $k \in \mathbb{N}$, the function $\mathbb{P}^{\,\eta^k}_{M,\mathbb{P}_{\mathsf{guess}}(\eta,-)}(\{0,1\}, P)$ be negligible. Also note that there is no need for r to be a constant, but it could be a function of η or any other parameter; the attacker, however can use the rule only if has enough resources, i.e. if $t \geq r$.

All the mentioned extensions do not significatively affect our results; in particular, Theorem 1 still holds in a model with all the above extensions, provided that the probability to break the cryptosystem is negligible. The following extensions might instead have an impact on our model.

Average–case vs. Worst–case Analysis. In the definition of $\mathbb{P}^{\,t}_{M}(\mathcal{K}, P)$, equation (3) only considers the transition which is the best one for the attacker (i.e., the worst case for P). Thus, it performs a *worst–case* analysis. An alternative would be an *average–case* analysis: we could consider all the transitions and weight each according to a given distribution. The hard point is to define a distribution that faithfully reflects both the scheduling of the concurrent actions by the protocol and the choices the attacker makes among its operations.

Random Choice. Sometimes protocols are specified using a toss–a–coin operation. In the computational approach this is easy, while the usual non–deterministic operator + of other process calculi is not adequate. A probabilistic choice operator $+_p$ (see e.g. Larsen and Skou [8]) accommodates well in our framework, originating transitions on the form $\xrightarrow[p]{\mu}$. This could be a first step towards

using formal methods for proving security properties, e.g. the correctness of zero-knowledge protocols, that have been faced so far only within the computational approach.

Behavioural Equivalence. Some authors advocate the use of *behavioural equivalence* for security properties, especially non–interference. They typically assume a standard Dolev–Yao attacker. It would be interesting to study if and how these notions change when a $\mathsf{DY}_{\mathbb{P}}$ attacker is assumed, instead.

A first step is by Abadi and Jürjens [3] who relate formal models with the computational model following [4]. They show that if two systems are equivalent from the formal point of view, then it is computationally hard for an eavesdropper to distinguish between them. (Note that they assume a passive attacker, unlike us.) Another quite different approach to probabilistic non–interference is in [12].

Partial Information. In the real world, the attacker may not be able to guess keys but still be able to perform statistical attacks and gather some partial information from intercepted messages. Unlike the computational complexity model, ours is not apt to study this kind of attacks. This is because we use a set to represent the knowledge of the attacker, thus implicitly assuming that the attacker either has complete knowledge of a term or no knowledge at all. In [7], Clark, Hunt, and Malacaria deal with information leakage using Shannon's entropy. Whether this approach can also be applied to process calculi and formal methods is still an open issue.

Beyond Secrecy. In our model we focus only on secrecy. However, the model could be extended to include other security properties such as, for instance, authentication properties. We think that the model could be generalized to address any *safety* property without much trouble. This could be the direction for future research.

Towards Adaptivity. In Sect. 3 we assumed that the guessing probability depends only on η, and therefore that we can express it as $\mathbb{P}_{\mathsf{guess}}(\eta)$. In the real world, however, it may depend on other factors. Most importantly, as time passes and the attacker interacts with the protocol, we expect the guessing probability to increase since the attacker might learn something from, for instance, its previous *guessing* attempts, even if they failed (thus it is an adaptive attacker).

One could then refine our model allowing $\mathbb{P}_{\mathsf{guess}}$ to increase as time passes, and thus express the guessing probability as $\mathbb{P}_{\mathsf{guess}}(\eta, \mathsf{time})$, where time is the current time. Also here our main theorem should hold, under a mild hypothesis on the growth of $\mathbb{P}_{\mathsf{guess}}$.

6 Conclusions

We presented a simple process calculus that can be used to specify cryptographic protocols. We introduced two distinct notions of secrecy, one borrowed from formal methods (DY–safety) and one adapted from the computational complexity

theory ($\mathsf{DY}_{\mathbb{P}}$–safety). Under suitable assumptions, we proved the equivalence of these two secrecy definitions.

A consequence of this result is that the perfect encryption assumption is quite acceptable. Thus, one can use the standard techniques for protocol analysis based on formal methods (e.g. type systems [1], control flow analysis [6], model checkers [9,11], etc.) to check if a given protocol is $\mathsf{DY}_{\mathbb{P}}$–safe.

Acknowledgements. Partially supported by the European Union FET project DEGAS, IST-2001-32072.

References

1. Martín Abadi. Secrecy by typing in security protocols. *Journal of the ACM*, 5(46):18–36, September 1999.
2. Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The Spi calculus. *Journal of Information and Computation*, 148(1):1–70, 1999.
3. Martín Abadi and Jan Jürjens. Formal eavesdropping and its computational interpretation. *Lecture Notes in Computer Science*, 2215:82, 2001.
4. Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP TCS2000*, Sendai, Japan, 2000. Springer-Verlag, Berlin Germany.
5. Michael Backes and Christian Jacobi. Cryptographically sound and machine-assisted verification of security protocols. In *Proceedings of 20th International Symposium on Theoretical Aspects of Computer Science, STACS '03*, pages 675–686. Springer-Verlag, 2003.
6. Chiara Bodei, Pierpaolo Degano, Flemming Nielson, and Hanne Riis Nielson. Static analysis for the π-calculus with application to security. *Information and Computation*, 168, 2001.
7. David Clark, Sebastian Hunt, and Pasquale Malacaria. Quantitative analysis of the leakage of confidential data. In Alessandra Di Pierro and Herbert Wiklicky, editors, *ENTCS*, volume 59. Elsevier, 2002.
8. Kim G. Larsen and Arne Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
9. J. K. Millen. The interrogator: A tool for cryptographic protocol security. In *Proceedings of the 1984 Symposium on Security and Privacy (SSP '84)*, pages 134–141, Los Angeles, Ca., USA, April 1990. IEEE Computer Society Press.
10. Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, May 1999.
11. J. C. Mitchell, M. Mitchell, and U. Stern. Automated analysis of cryptographic protocols using murϕ. In *Proceedings of the 1997 Conference on Security and Privacy*, pages 141–153. IEEE Computer Society Press, 1997.
12. Alessandra Di Pierro, Chris Hankin, and Herbert Wiklicky. Approximate non–interference. In *Proceedings of CSFW'02 – 15th IEEE Computer Security Foundations Workshop*, Cape Breton, Nova Scotia, Canada, 2002.
13. Angelo Troina, Alessandro Aldini, and Roberto Gorrieri. A probabilistic formulation of imperfect cryptography. In N. Busi, R. Gorrieri, and F. Martinelli, editors, *Proceedings of Int. Workshop on Issues in Security and Petri Nets (WISP'03)*, pages 41–55, June 2003.