Guessing Attacks in the pi-calculus with a Computational Justification. Draft Comments Welcome.

Tom Chothia
tomc@lix.polytechnique.fr
Laboratoire d'Informatique (LIX)
École Polytechnique (CNRS)
91128 Palaiseau Cedex - France

April 5, 2005

Abstract

This paper presents an extension of the pi-calculus that can reason about brute force and guessing attacks. We relate new name declarations in the pi-calculus with random sampling in the computational model of security. The scope of a new name can then be expanded at a comparable cost as it would take to guess the randomly sampled value in the computational setting. We provide a syntax and reduction semantics for this system and a function that calculates the cost of a given attack, taking into account the ease with which the attacker can confirm their guesses.

We argue the correctness of this calculus by relating it to the computational model of security. We show that if the cost of an attack in the calculus is less than exponential in a security parameter, then there exists a polynomial time Turing machine that can defeat the process with non-negligibility probability. On the other hand, if there is no sub-exponential cost attack, then the process is just as safe as its spi-calculus counterpart, and so the use of guessable names does not help the attacker.

1 Introduction

Formal models [DY83, Low96, BAN96, AG97, THG99, Cer01, FA03] can check the security of protocols in a powerful yet simple way. However, these models often miss faults based on computational factors which in turn can be reasoned about by much more complex, computational models of security [GM84, BM84, GGM86, GMR88, BKR00].

This paper is primarily concerned with guessing attacks on protocols. Randomly guessing values can be an effective way to break a protocol. However, guessing attacks are handled poorly by most formal security analysis techniques, including the spi-calculus [AG97] with its model based on free and bound names. Our aim in this paper is to provide a pi-calculus [Mil91] based model of guessing attacks with most of the simplicity of the formal model and some of the power of the computational model.

In Section 2, we review the formal and computational foundations on which this paper builds. The spi-calculus based security analysis method [AG97] use a mini-language with a construct to create new names. These new names can never be guessed, so an attacker must trick a process into giving away its secrets. The computational security model, on the other hand, is based on the random sampling of key values followed by a complexity analysis of the run time of any possible attacker.

In assembling our calculus we equate the new name declaration of the pi-calculus and random sampling in the computational model. In our calculus we write *new* $a : D_n$ to mean that a is a new name, randomly sampled from a domain of size n.

To avoid the difficult complexity analysis required by the computational model we only allow the attacker to take advantage of the limited domain size through a guess operation: *guess* $x : D_n$. This operation allows an attacker to correctly guess a value from a domain, but at a cost. An attacking process can use this *guess* operation to try to break a protocol. The attackers trace maps out a path to be followed later by another brute force attack. A real attacker would have to use trial and error to guess values, look for confirmations of their guesses and be prepared to go back and guess again. We can calculate the cost of this more complicated guessing attack from a single trace of a successful attack in the calculus using a simple function on traces. This function can find the cost of multi-

ple guesses, taking into account exactly when the attacker can get their guesses confirmed. For a given protocol, the higher the value of the minimum cost attack the safer the protocol will be.

One use of this calculus is to find weak points in protocols. For instance, a good banking protocol might force the attacker to guess the victims PIN number and account number at the same time, at a cost of $10^4 * 10^{12}$ where as a less secure protocol might allow the attack to verify the pin number from the bank card before it has to guess the account number, thus making the cost of an attack $10^4 + 10^{12}$. In most models both these protocol would be considered safe and indistinguishable. The calculus we present here could tell the two processes apart by the minimum cost of a successful attack on each.

1.1 Contributions

We show the correctness of the calculus and the cost function by relating our model back to the spi-calculus and to the computational model. First we add a security parameter for domain sizes. We define a Turing machine attacker that can interact with a process. In our calculus we translate new names as random bit strings chosen from a domain of size equal to the new name's domain size, whereas for the spi-calculus all names are mapped bit strings with the same length as the security functions, hence they cannot be guessed in sub-exponential time. We also give a safety criterion for these Turing machine attackers. For our first theorem, we show that if there is a sub-exponential cost attack in our calculus then the process is unsafe in the computational model, i.e. there exists a polynomial time Turing machine that can defeat the safety criterion.

We observe that zero cost attacks in our calculus correspond to attacks in the spi-calculus but given that we have changed the spi-calculus model of security we want to be sure that we have not allowed new attacks that fall outside of our model. Work such as [AR00, BPW03, Lau04, MW04, War03] has proved that there is a computationally correct mapping from the Dolev-Yao formal model [DY83] to the computational model.

Our second theorem proves that if there exists a correct mapping for the spi-calculus, then there is also a correct mapping for our calculus. This result shows that in shortening some of the new names in the guessing calculus, we do not allow for any new attacks that cannot be found in the calculus itself.

It should be noted that a proof of correctness does not mean that there is absolutely no possibility of an attack on a protocol, as the computational model is itself a model, just a somewhat broader one than the pi-calculus. It does not, for instance, consider attacks based on time or power usage. For one example of a computational correct encryption algorithms falling to such attacks see [Koc96].

Protocols can be made safer by taking measures to counter guessing attacks. For instance, the French debit card system will lock a card after receiving three incorrect PIN numbers. These precautions are often effective, however they can also be inconvenient, (three wrong tries at a PIN number is not inconceivable for someone with a new bank card). The introduction of these safety measures will also complicate any protocol and could in turn lead to new security flaws. It would be better to design the protocol well in the first place to make guess attacks harder without these counter measures.

1.2 Related Work

Zunino and Degano [ZD04] enhance the standard Dolev-Yao attacker so that it can guess a key. Using computational security methods they show that there is a negligible probability of these kind of guesses succeeding and so standard Dolev-Yao attacker is just as powerful as the enhanced one.

Mitchell, Scedrov et al. [MRST01, RMST04, MMS03] extended th pi-calculus with polynomial time functions. The calculus is then refined by placing a polynomial bound on the number of times a replicating process can replicate and by limiting the capacity of each channel to some polynomial value. The result of this is a process calculus that can express complex numerical attacks but is guaranteed to run in polynomial time. Therefore, if a protocol can be broken in this calculus, we know the protocol can also be broken by a polynomial Turing machine. A key difference between the PPC and the work we present here, is that the PPC limits the non-determinism of a processes to ensure that it runs in polynomial time, whereas we allow any non-deterministic pi-calculus process but we do not allow these processes to guess values, except by explicit use of a guess operation.

There is a great deal of work on the subject of attacks based on guessing poorly chosen passwords, for instance Gong et al. [GLNS93]. This work focuses on the idea that passwords are often chosen poorly and so can sometimes be guessed by dictionary attacks. Much of this work considers decryption functions to simply map bit strings to bit strings and so, in order to verify the guess of a key from an encrypted message you must have access to that encrypted message and know the message's contents. Lowe [Low02] works in the Dolev-Yao model and analysis protocol in the FDR model checker. He shows how

some protocols might be vulnerable to attack and how these protocols can be made safer. Delaune and Jacquemard [DJ04] show that, for active attackers, this type of guessing attack can be found in polynomial time. Corin, Doumen and Etalle present similar work in the Applied pi-calculus [CDE03] and provide a tool to find these attacks [CMAFE03].

Multi-set rewriting [Cer01] is another formal analysis system for checking protocols. It has been extended with a simple notion of additive cost on actions. This system could be a useful base from which to investigate guessing attacks.

There is a growing body of work that looks at bridging the gap between the computational computer security model and formal analysis. We discuss this work after introducing the computational security model in the next section.

In Section 2, we review some background work, including the pi and spi-calculi and the computational security model. Section 3 introduces the pi-calculus with guessing, and the cost function for attacks is giving in Subsection 3.4. We address the correctness of our system in Section 4 and finally we conclude and discuss further work in Section 5.

2 Background: Formal and Computational Analysis

This paper is aimed at combining some aspects of computational analysis methods with formal process analysis in the pi-calculus. So, this section first reviews protocol analysis in the pi and spi-calculi and then outlines how computational methods can be used to prove much stronger results. We also, briefly, discuss some related work that falls into the gap between these two fields.

2.1 The pi and spi-calculus

The pi-calculus is a miniature concurrent language. This language is simple enough to allow formal analysis while expressive enough to describe most interesting concurrent processes. The exact syntax given to the pi-calculus varies from paper to paper; here we use the following:

$$\begin{array}{lcl} \text{Process } P,Q & ::= & 0 \\ & \mid & \mathit{send}\ a\langle b\rangle \\ & \mid & \mathit{rec}\ a(x);P \\ & \mid & \mathit{new}\ a;P \\ & \mid & !P \\ & \mid & (\ P \mid Q\ ) \\ & \mid & [a=b];P \end{array}$$

The first piece of syntax 0, represents the stopped process. The *send* operation broadcasts the name b over channel a. The next operation receives a name over the channel a and substitutes it for x in the continuing process P. The *new* operation creates a new communication channel that is guaranteed to be new and unique. The bang operator ! can perform recursion by spinning off an arbitrary number of copies of a process. Placing a compulsory input guard on this command avoids unintentionally spinning off an infinite number of processes. The bar | represents two processes running in parallel and finally the match operation, $[a=b];P$ executes P if and only if a is equal to b.

The key reduction rule of the calculus allows two processes to communicate:

$$\mathit{send}\ a\langle b\rangle \mid \mathit{rec}\ a(x);P \longrightarrow P[b/x]$$

To avoid enumerating every possible arrangement of processes in the semantics rules, the pi-calculi often use a structural equivalence relation to identify some processes as the same, for instance we say that $P \mid Q \equiv Q \mid P$. A single semantic rule then extends a reduction of one process to all other equivalent processes.

$$\frac{P \equiv P_1 \quad P_1 \rightarrow Q_1 \quad Q_1 \equiv Q}{P \rightarrow Q}$$

One of the most important aspects of the pi-calculus is that new names are both new and unguessable, for instance the process $\mathit{new}\ a;\mathit{rec}\ a(x);P$ can never receive a communication on the channel a, no matter what the surrounding processes might try. This also means that there is no way to write down an attack that tries every possible value until it "finds" the right one.

In order to model a larger number of interesting protocols the spi-calculus extends the pi-calculus with primitives for encryption. A new term is added, of the form $\{M\}_N$, to mean the message M encrypted with the key N. These terms can be decrypted using an operation of the form: $\mathit{case}\ L\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P$. If a process provides the correct key, the semantics rule for decryption substitutes the encrypted value for the variable:

$$\mathit{case}\ \{M\}_N\ \mathit{of}\ \{x\}_N\ \mathit{in}\ P \longrightarrow P[M/x]$$

Encryption of the message M with the key N is performed by simply writing the term $\{M\}_N$.

2.1.1 Security Analysis in the spi-calculus

The small size and expressivity of the spi-calculus makes the detailed analysis of processes possible. This including

the analysis of security properties, as shown by Abadi and Gordon [AG97]. We briefly summarize some of this work here.

An attacker is modelled as a *context*, written $C[_]$ which is any possible surrounding term, into which the process being attacked may be plugged. This means that the attacker can access any of the process's free names (those not bound by a new or input operator) and use these to communicate with the process in anyway it chooses. The attacker may also perform any computations it wishes using the spi-calculus.

The barbs of a process are the unguarded outputs. We write $P \downarrow a$ to means that P has the barb a.

$$send\ a\ (\vec{b}) \downarrow a \qquad \frac{P \downarrow a}{P \mid Q \downarrow a}$$

$$\frac{P \downarrow a \quad a \neq b}{new\ b; P \downarrow a} \qquad \frac{P \downarrow a \quad P \equiv Q}{Q \downarrow a}$$

We also write $\Downarrow$ for $\downarrow$ closed under reduction. A test then consists of a testing process R and an action a and we can use this test to define an equivalence:

Definition 2.1 (Testing Equivalence) *$P \simeq Q$ if for any test (R,a) we that have $(P \mid R) \Downarrow a$ if and only if $(Q \mid R) \Downarrow a$*

To prove a value inside a process is secure we prove that the process is testing equivalent to itself for any possible, different values. For example, consider the following system:

$$\begin{array}{rcl} System(M) & \equiv & new\ pwd; (\ Client \mid Server(M)\ ) \\ Client & \equiv & new\ reply; send\ ser\langle pwd, reply\rangle \\ & & \mid rec\ reply\ x; P(x) \\ Server(M) & \equiv & !rec\ ser\ (x,y); [x = pwd]; send\ y\ \langle\ M\ \rangle \end{array}$$

This system is comprised of a client and a server. The server listens for a connection on a public channel, this connection is made up of a password and a reply channel. The server then checks the password and, if it is correct, replies with the message M. The password is private between the client and the server, therefore no attacker may know, or guess, it and hence the message M is safe.

We could prove this formally by showing that the for any M and M', if $P(M) \simeq P(M')$ then the system $System(M) \simeq System(M')$ which would in turn imply that for all hostile attackers A, the process $A[Systems(M)] \simeq A[System(M')]$ and so P cannot leak the message M to the attacker A.

2.2 The Computational Security Model

Implementations of formal processes are susceptible to attack that are "outside their model". For example, a new name that is implemented as a bit can be correctly guessed with a one in two chance. Whereas, the spi-calculus model of the process might say that this value is a bound name and hence is guaranteed to be unguessable (alternatively the name could be free, then the attacker would always know it).

The computational model of security avoids some of these problems by modelling attackers as polynomial Turing machines and values as bit strings [GM84, BM84, GGM86, GMR88]. This means that the attacker can carry out any computational feasible operation. Secret values in this model are randomly chosen from a probability distribution. Given a probability distribution D_n of size n, we write:

$$x \stackrel{r}{\leftarrow} D_n$$

to mean that x is a value that has been randomly sampled from that distribution.

A security criterion is used to judge the safety of a given process. The choice of which criterion to use will depend on the exact nature of the required security property. However, a typical criterion defines the *attackers advantage* to equal the probability that, for some sampled value, the attacker can correctly identify that value, minus the probability that the attacker incorrectly identifies the value.

Attackers are probabilistic, polynomial time Turing machines (PRTMs), this is a Turing machine that runs in polynomial time in some security parameter and has the ability to make a random choice. It is enough for these attackers to return 0 or 1 depending on what they believe to be the result of their attack.

The chance of the attacker defeating the criterion must become very small, very quickly as the size of the security parameter grows. More formally, we say that the attackers advantage be *negligible*.

Definition 2.2 (negligible) *A function f is negligible if for all c there exists N such that for all $x > N$ we have that $f(x) < x^{-c}$.*

As an example of a simple criterion, one could say that an encryption scheme E is safe if

$$\begin{array}{rcl} Adv(n) & = & Pr[x \stackrel{r}{\leftarrow} D_n, k \stackrel{r}{\leftarrow} Key_n : A(n, E_k(x), x) = 1] \\ & & - Pr[x, y \stackrel{r}{\leftarrow} D_n, k \stackrel{r}{\leftarrow} Key_n : A(n, E_k(y), x) = 1] \end{array}$$

is negligible. In this criterion. The attacker is given the length of the security parameter, a random element and an

encrypted value. The attacker must answer 1 if it believes that the value it has been giving is the same as the encrypted value and 0 if it believes the values are different.

A stronger, and more realistic criterion might give the attacker the ability to use the encryption algorithm, this is done in the form of an *oracle* and we write $A^{f(\text{-})}$ to mean the attacker has access to an oracle to perform the function f with its chosen input. So, for instance, a criterion to ensure that the attacker does not know when the same value has been encrypted a number of times could be written as

$$\begin{aligned} Adv(n) \;=\; & Pr[k \xleftarrow{r} Key_n : A^{E_k(-)}(n) = 1] \\ & - Pr[k \xleftarrow{r} Key_n, x \xleftarrow{r} D : A^{E_k(x)}(n) = 1] \end{aligned}$$

where the false oracle $E_k(x)$ ignores the value the attacker gave it to encrypt and always returns the encryption of x with the key k.

Whereas, a criterion that require key identities to be concealed could be written as:

$$\begin{aligned} Adv(n) \;=\; & Pr[k,k' \xleftarrow{r} Keys_n : A^{E_k(-),E_{k'}(-)}(n) = 1] \\ & - Pr[k \xleftarrow{r} Keys_n, x \xleftarrow{r} D : A^{E_k(-),E_k(-)}(n) = 1] \end{aligned}$$

The computational method captures, and hence defends against, a wide range of possible attacks, including guessing attacks. However, proofs in this model are none trivial, and a successful proof of correctness can be the basis for a published paper [BKR94]. The aim of our work is to provide some of the benefits of computational analysis of guessing attacks without involving the user in any difficult proofs.

2.3 Bridging the gap

There have already been a number of papers that bridge the gap between formal and computational analysis. However, most of these are based on finding conditions under which you can derive a computational proof from a formal analysis.

Hüttel shows that the spi-calculus is Turing powerful [Hüt02]. A key paper, which set the groundwork for others, is by Abadi and Rogaway [AR00]. They define a translation from formal Dolev-Yao [DY83] model terms into computational terms: key terms in Dolev-Yao become newly generated keys. Nonces become new, hard to guess strings. Dolev-Yao equivalence then implies computational indistinguishability.

This relation between the Dolev-Yao model and the computational model has been greatly expanded by Backes, Pfitzmann and Waidner who, amongst other things, deal with active attackers and nested encryption [BPW03] and symmetric encryption [BP04b]. By using this link between formal and computational secuirty, they also find correctness proofs of the Needham-Schroeder-Lowe and Otway-Rees Protocols [BP04a, Bac04]. Micciancio and Warinschi [MW04] observe that Abadi and Rogaway's mapping is not complete, but can be made complete by using a stronger security criterion. Janvier, Lakhnech and Mazare [JML05] show that Dolev-Yao is computationally sound in the presence of active attackers. Hezrog [Her04] shows correctness for a stronger definition of non-malleability for Dolev-Yao. This provides security against adaptive attackers. He also shows that the correctness of Dolev-Yao is preserved when it is extended to include some kinds of new operators, included those needed for Diffie-Hellman.

3 The pi-calculus with Guessing

This section introduces the pi-calculus extended with guessing, the pi-g calculus. We do this in a number of stages, in the hope of illustrating the motivation behind the design decisions and elucidating some of the finer details. The first stage introduces the guess action, next we consider the cost of multiple guesses, and thirdly we show that the cost of an attack can be reduced if the attacker can get easy confirmation of their guess. Finally, note that only certain kinds of actions give reliable confirmations and we develop our cost function accordingly. The full calculus is given at the end of this section and a computational justification for the informal reasoning is given in Section 4.

3.1 A Guessing Rule

Given a protocol with a fixed password, there is a subtle difference between finding a general guessing attack on the protocol and an attack that guesses the password for one particular run of the protocol. To illustrate this point consider Lowe's attack on the Needham-Schroeder Protocol [Low96]. A rough, back of the envelope calculation finds that this attack happens over 14 steps with around 75-256 possible, interesting messages for the attacker at each step. This makes the chance of finding this attack, at random, to be less than 1 in 2^{100}. If the nonce used in this protocol was 64 bits long, how can we tell our automated protocol checker not to attack the protocol by guessing the nonce directly?

The pi-calculus handles this quite neatly, by using a new name for the secret value, for instance, a system in which processes P and Q share a password pwd against an attacker A could be written as:

$$new\ pwd; (\ P \mid Q\ ) \mid A$$

This marks out the password as important to the correctness of the process and ensures that the attack can never come up with the name without being told. However, this can sometimes entrust too much security in the new name; a password consisting of a single bit, for instance, can be easily guessed.

As mentioned above, the computational model randomly samples these names. In this setting we would write:

$$pwd \xleftarrow{r} D_n \ : \ P(pwd) \mid Q(pwd) \mid A$$

This stops the attacker from just knowing the password at the start of the attack because the same attack must work for different, randomly sampled values of *pwd*. This model does allow for a brute force attack, however it forces anyone using this method to perform a statistical analysis on the run time complexity of A.

To bring these two methods closer we work in the pi-calculus and allow a new name to be sampled from a domain of a given size, writing

$$new\ pwd : D_n; (\ P \mid Q\ ) \mid A$$

to mean *pwd* is a new value, shared between P and Q, sampled from a domain of size n. To avoid the need to do nasty analysis we force the attacker to declare when it is attempting to guess a value and pay a cost proportional to the size of the domain. The key rule being

$$\begin{array}{l} new\ pwd : D_n; P \mid guess\ x : D_n; A \\ \qquad \rightarrow_{pwd:n} new\ pwd : D_n; (P \mid A[g_{pwd}/x]) \end{array}$$

which means that the attacker guesses the name *pwd* at a cost of n.

The distinct name g_{pwd} is a correct guess of the name *pwd*. We do not substitute *pwd* for x directly as we will later need to find out when a guess is confirmed. This label on the reduction, as with all the other labels records information necessary calcule the cost of the attack, no label ever affects the reduction of a process. A new name declared of the form $new\ a : D_n$ binds both a and g_a, furthermore we only allow names of the form g_a to be declared with the *guess* operator.

So, whereas before, in the pi-calculus, an attack could only acquire knowledge of a new name by being told, the attacker can now also use the guess action and pay the cost of a brute force attack on a value of the given domain size.

3.2 Multiple Guesses

Now we have allowed the attacker to make a single guess we must calculate the cost of multiple guesses. Below we give a simple system in which a process shares a 128 bit password with another process Q. An attacker A will try to guess this password one bit at a time.

$$\begin{array}{l} Process \equiv new\ chn, rew, b_1 : D_2 ... b_{128} : D_2; \\ \quad (send\ a\langle chn, rew\rangle \mid rec\ chn(x_1); [x_1 = b_1]; ... \\ \quad ... rec\ chn(x_{128}); [x_{128} = b_n]; send\ rew\langle reward\rangle) \\ \quad \mid Q\ ) \end{array}$$

$$\begin{array}{l} Attacker \equiv rec\ a(chn, rew); \\ \qquad (!(guess\ b : D_2; send\ chn\langle b\rangle) \\ \qquad \quad \mid rec\ rew(x)) \end{array}$$

The password process waits for a connection, and replies to this with a new, secure channel. P then listens on this new channel for the bits of the password. One by one, the received bits are tested against the bits of the true password and if they all match the *reward* is broadcast.

The attacker A cannot know the bits of the password because they are new names bound to P and Q. However, it can guess each bit with a cost of 2 (or a 1 in 2 chance of being correct). After the attacker has made a guess at all 128 bits of the password it will have a 1 in 2^{128} chance of having correctly guessed the password and hence the cost of these guesses should be 2^{128} So, in this case, we multiply the cost of each successive guess.

It should be noted that the process A does not perform an attack of cost 2^{128} on the process P, rather 2^{128} is the cost of performing a successful attack along the lines of the attack attempted by A.

3.3 Conformation of a Guess

Multiplying the costs of all guesses in a trace will sometimes overestimate the cost of an attack. The multiplication of the costs reflects the idea that all possible combinations of guesses must be tried. However, if the attacker can get one of their guesses confirmed as correct, when only part of the way through the attack, then the confirmed guess will not have to be retried and will therefore not contribute to the cost of future guesses.

For example, consider the fairly moronic extension to our prior password system given below. The processes function in the same way as before, except this time an acknowledgment is broadcast after receiving each bit.

$$\begin{array}{l} Process \equiv \\ new\ chn, rew, b_1 : D_2 ... b_{128} : D_2; \end{array}$$

$$(send\ a\langle chn, rew\rangle \mid rec\ chn(x_1); [x_1 = b_1]; (\text{send ack} \mid \ldots$$
$$\ldots rec\ chn(x_{128}); [x_{128} = b_{128}];$$
$$(\text{send ack} \mid send\ rew\langle reward\rangle)\ldots)$$
$$\mid Q)$$

Because the system tests each bit before listening for the next bit these acknowledgments confirm the guess of the previous bit as correct. This means that any brute force attack based on guessing and then listening for an acknowledgement would only have to try a guess at each bit once making the total cost for the new attacker $128+1$.

These confirmations effectively prune the search tree of all the possible combinations of guesses. Figure 1 shows the tree of all possible guesses for the original password protected processes and the pruned tree for the password processes with acknowledgments signals.

To allow dependences to be tracked we extend the syntax of the calculus with a dependence marker. This marker takes the form: $(g_a)P$ and behaves in exactly the same way as the processes P. The point of the marker is to record that the process P is only running because the guess g_a has been shown to equal a. The key semantic rule for this is:

$$[g_a = a]; P \rightarrow_{(a)} (g_a)P$$

The label (a) signals that this action build a dependence on the guess being correct.

A guess is confirmed when a signal that depends on the guess being correct passes from the process to the attacker. So, we make the attacker and victim explicit by separating them using a double bar $||$. The following reductions illustrate both these mechanisms:

$$\begin{array}{ll} & new\ a : D_n;\ rec\ b(x); [x = a]; send\ b\langle c\rangle \\ & \quad || \ guess\ y : D_n; (send\ b\langle y\rangle \mid rec\ b(x)) \\ \rightarrow_{a:n} & new\ a : D_n;\ (\ rec\ b(x); [x = a]; send\ b\langle c\rangle \\ & \quad || \ send\ b\langle g_a\rangle \mid rec\ b(x)) \\ \rightarrow & new\ a : D_n;\ ([g_a = a]; send\ b\langle c\rangle \ || \ rec\ b(x)) \\ \rightarrow_{(a)} & new\ a : D_n; (\ (g_a) send\ b\langle c\rangle \ || \ rec\ b(x).A\ ) \\ \rightarrow_{a:\overline{b}\langle\vec{c}\rangle} & new\ a : D_n;\ (\ 0 \ || \ A\ ) \end{array}$$

The name a, from a domain of size n, is only known by the process on the left hand side of the double bar. In the first reduction the attacker guesses this name, hence the reduction is labelled with $a : n$ to indicate that the name a has been guessed at cost n. The second step is a communication between the attacker and the process. Now, we come to the match $[g_a = a]$. This match will go ahead as, for the sake of computing the cost, we consider the guesses to be correct. However, we must leave a dependence marker to say that the resulting process is only running because a guess has been shown to be correct. In the final step another communication happens and as the *send* is dependent on the guess of a and the communication takes place between the main process and the attacker, the guess of the name a is possibly confirmed by the communication of $\vec{c}$ over the channel b, indicated here by the $a : \overline{b}\langle\vec{c}\rangle$ label on the reduction. It is left to the cost function to work out if the confirmation is reliable This results in the rule for conformations:

$$(\overrightarrow{g_a}) send\ b\langle\vec{c}\rangle \ || \ rec\ b(\vec{x})A \rightarrow_{(\vec{a}:\overline{b}\langle\vec{c}\rangle)} P \ || \ A[b/x]$$

As the dependence marker can be passed from one part of the system to another (see the first rule in Figure 3) this also catches indirect communication comfirmation.

The dependence markers are added to the calculus purely to aide the analysis of a process. They would not be present in any implemented system. When an attacker receives a message from an action of the form $(g_a)send\ a\langle b\rangle$ they only see the $send\ a\langle b\rangle$ part. So, if along with a confirming signal there are other, similar signals that do not confirm a guess, the attacker cannot be sure which one they have received and so the signal does not definitely show the guess to be correct. As an example, we ask the reader to again consider the password process given above. There, a correct guess is confirmed by a signal on the channel *ack*. However, if we added a continually repeating output on the channel *ack*, i.e. a process of the form $send\ loop \mid !rec\ loop; send\ ack$, then the receipt of an *ack* signal by the attacker would be meaningless and would not help him prune down the search tree of possible guesses. So, when the cost function is presented with a possible confirmation it uses the state of the process, recorded from when the dependence was first built, to test if the confirming action would be possible without the guess being correct.

3.4 Cost of a Trace

When an attacker finds a guessing attack for a protocol, the attacking process does not have to model the ways in which it may try different guesses and keep track of the guesses already made. This would lead to a more complicated system, which would require the kinds of analysis seen in the computational model. Instead, we restrict the only type of computation that can be used to find secret names to the *guess* action. We can then calculate the cost

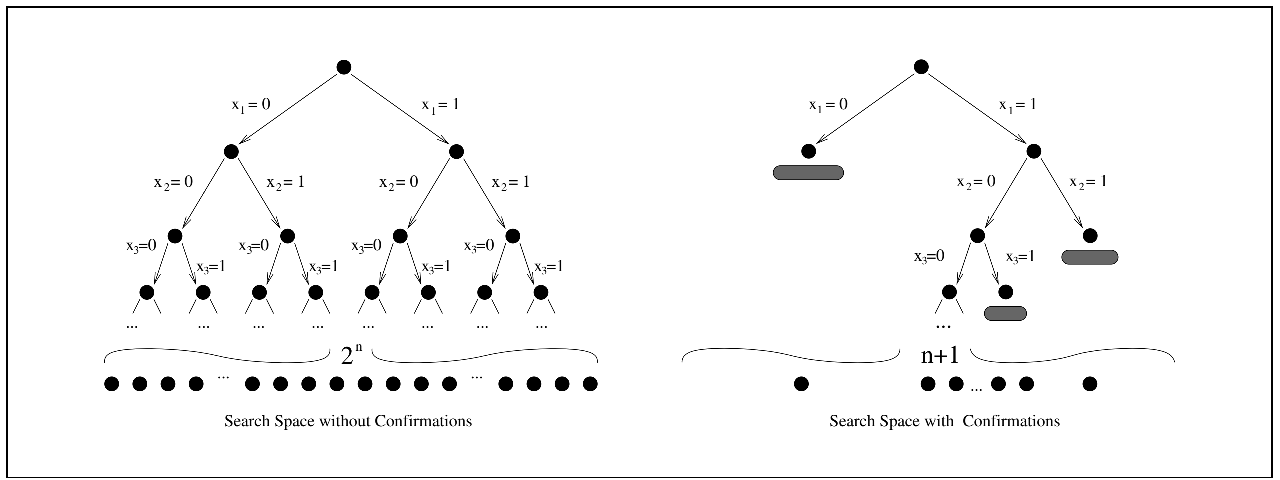

Figure 1: The Possible Guess of all Values, with and without Confirmations

of these guess actions separately. It should be noted that we do not calculate the chance of the attacking process being correct. Rather, our cost function calculates the cost in terms of computing power or number of tries, a brute force attacker would need to successfully follow the same path as the attacking pi-g calculus process. The cost of guesses are calculated and added to the total as they are confirmed and the cost of the unconfirmed guesses is calculated at the end of the trace. We calculate the cost of a trace, defined by the grammar $T ::= P \rightarrow_\alpha T|P$, using an auxiliary function that has a list of current guesses and another list that stores the states in which the dependences where first build.

$$cost\ of\ trace(\ T\ ) = cost(\ T, [], []\ )$$

The *cost* function has five cases. The first is that of a guess being made by the attacker.

$$cost(P \rightarrow_{a:n} T, gu, comf) = cost(T,\ (a,n);gu, comf)$$

The guess and its cost are added to the list of current guesses, *gu*. In the second case a dependence on a guess being correct is built up inside the protocol and we add the first processes state to the *comf* list.

$$\begin{array}{ll} cost(P \rightarrow_{(a)} T, gu, comf) = & \\ \quad cost(T,\ gu,\ (a:P);comf) & if\ (a:Q) \notin comf \\ \quad cost(T,\ gu, comf) & if\ (g_a:Q) \in comf \end{array}$$

The confirmation rule is the most complicated. To simplify matters we introduce a short hand for the produce of the costs (the second elements) of the list of current guesses: $\Pi\ snd\ [(a_1,n_1),...(a_i,n_i)] = n_1 \times n_2 \times ... \times n_i$. The action tag $(\vec{a}, \overline{b}\langle\vec{c}\rangle)$ tells us that the elements of the set $\vec{g_a}$ are potentially being confirmed by a communication on the names $\vec{c}$ over the channel b. We then check in the confirmation list to see if the system could have made the same communication if the each guess had been wrong, if it could not have then the confirmation is real.

$$\begin{array}{l} cost(P \rightarrow_{(\vec{a}, \overline{b}\langle\vec{c}\rangle)} T, gu, comf) = \\ \Pi\ snd\ (gu[(a_i, m_i - 1)/(a_i, m_i)]) \\ \qquad + cost(T, gu \backslash \{(a_1, m_1)..., a_j, m_j)\}, comf) \end{array}$$

where $a_1, ..., a_j$ are all the names in $\vec{a}$ such that $(a_i, Q) \in comf$ and $(a_i, m_i) \in gu$ and $new\ d; Q\{d/g_{a_i}\} \not\Rightarrow C[send\ b\langle\hat{c}\rangle]$

If the attacker knows the contents of the message c, the attacker can use this to look for confirmations, so $\hat{c} = \vec{c}$. However, if any of $\vec{c}$ are secret names within the protocol then the attacker will not be able to distinguish it from any other secret name from the same domain. So that element of $\hat{c}$ can be any bound name that is unknown to the attacker and is drawn from the same domain.

The cost produce by the confirmation is the produce of the costs of each guess, with one subtracted from all the confirmed guesses (as one path for each confirmed guess will continue) plus the cost of the rest of the trace, with the confirmed guesses removed from the list of current guesses. It would be possible for the confirmation action to come from a second dependency not from the state Q, however Q could reduce to the process that made this second dependence and so could perform the same actions.

It is unnecessary for this rule to confirm a guess of the messages contents. If the broadcast of this message

was dependent on a guess of c then the guess of c will be automatically confirmed. If the broadcast was not dependent on the guess then the attacker can read the value of c from this action and so does not have to guess it at all.

It would be quite possible for an attacker to make a guess and wait for an action they believe confirms a guess but in fact receive a reply from a different reduction that does not confirm the guess at all. For now, we say that it is unsafe for the security of the processes to depend on the scheduler avoiding these confirming states, but we will also give a computational argument for this in Section 4 that proves, that for any reasonable scheduler, this definition of a confirming state is enough to correctly find a process insecure.

For instance consider the process

$$[g_a = a] rec\ b; send\ c \mid send\ b \mid rec\ b; send\ c$$

This process will preform an output on c whether of not the guess is correct. If we gave the attacker full control over the oracle it would be able stop the communication between that does not confirm the guess. In which case, we would find a confirmation but we would not find a useful attack.

We also do not demand that the scheduler gives equal chance of performing each action, as an unrealistic assumption. This means that we do not accept confirmation from processes such as the following:

$$\begin{array}{l} [g_a = a]; rec\ b; (\ new c; (\ send\ c \mid \\ \quad rec\ c; send\ d \mid rec\ c; send\ d \mid rec\ c; send\ e\ ) \\ \mid send\ b \\ \mid rec\ b; (\ new c; (\ send\ c \mid \\ \quad rec\ c; send\ d \mid ; rec\ c; send\ e \mid rec\ c; send\ e\ ) \end{array}$$

Under a scheduler that gave equal chance of execution to each action, it could be augured that seeing a send on the channel d would tend to confirm the guess of a. While this may be interesting further work it is unclear if an attacker could rely on this kind of information so we leave it outside of our model, for now.

When we come to the end of a trace we multiply the remaining guesses to get the final cost.

$$cost(P, gu, comf) = \Pi\ snd\ gu$$

If a reduction does not fit into any of the above categories it has no effect on the cost, as shown by the last case.

$$cost(P \rightarrow T) = cost(T, gu, comf)$$

We summarise the cost function in Figure 2.

$$cost(P \rightarrow_{a:n} T, gu, comf) = cost(T,\ (a,n); gu, comf)$$

$$\begin{array}{ll} cost(P \rightarrow_{(a)} T, gu, comf) = & \\ \quad cost(T,\ gu,\ (a:P); comf) & if\ (a:Q) \notin comf \\ \quad cost(T,\ gu, comf) & if\ (g_a:Q) \in comf \end{array}$$

$$\begin{array}{l} cost(P \rightarrow_{\alpha} T, gu, comf) = \\ \quad \Pi\ snd\ (gu[(a_i, m_i - 1)/(a_i, m_i)]) \\ \quad\quad + cost(T, gu \backslash \{(a_1, m_1) ..., a_j, m_j)\}, comf) \end{array}$$

where if $\alpha = (\vec{a}, \overline{b}\langle\vec{c}\rangle)$ then $a_1, ..., a_j$ are all the names in $\vec{a}$ such that $(a_i, Q) \in comf$ and $(a_i, m_i) \in gu$ and $new\ d; Q\{d/g_{a_i}\} \not\Rightarrow C[send\ b\langle\hat{c}\rangle]$ and if $\alpha = (a, 0)$ then $a_1 = a$

$$cost(P, gu, comf) = \Pi\ snd\ gu$$

$$cost(P \rightarrow T) = cost(T, gu, comf)$$

Figure 2: The Cost of a Trace

3.5 Encryption

Encryption is an essential part of many interesting protocols. We can add encryption to our calculus in the manner of the spi-calculus, by adding the name $\{a\}_k$ to mean the name a encrypted with the key k, and adding the operator $decrypt\ a\ as\ \{x\}_k; P$ to decrypt, encrypted messages. Decryption adds another way in which a guess can be confirmed and dependences built up. Firstly, when a protocol uses decryption with a real key and a guess it sets up a dependence:

$$\begin{array}{l} decrypt\ \{b\}_a\ as\ \{x\}_{g_a}; P \rightarrow_{(a)} (g_a) P[b/x] \\ decrypt\ \{b\}_{g_a}\ as\ \{x\}_a; P \rightarrow_{(a)} (g_a) P[b/x] \end{array}$$

Secondly, an attacker's guess at a key can be confirmed directly by attempting to decrypt a message encrypted with the real key. This leads to a secondary decryption rule just for the attacker:

$$decrypt\ \{b\}_a\ as \{x\}_{g_a}; A \rightarrow_{(g_a, 0)} A[b/x]$$

The 0 indicates that this action always confirmation the guess.

The duel confirming decryption rule, where the message is encrypted with the guessed key and the attacker is using the real key, would not make sense, as it would mean that the attacker was trying to confirm a value they already know. As mentioned in the introduction, some formal definitions would not consider this a

$$(\overrightarrow{g_c})send\ a\langle\vec{b}\rangle \mid (\overrightarrow{g_d})rec\ a(\vec{x});P \rightarrow (\overrightarrow{g_c},\overrightarrow{g_d})P[b/x]$$

$$(\overrightarrow{g_a})send\ b\langle\vec{c}\rangle \mid\mid rec\ b(\vec{x})A \rightarrow_{(\vec{a}:\overline{b}\langle\vec{c}\rangle)} P \mid\mid A[b/x]$$

$$[a=a]P \rightarrow P \qquad\qquad [g_a=a]P \rightarrow_{g_a} (g_a)P$$

$$new\ a:D_n;(P \mid\mid guess\ x;A) \rightarrow_{a:n} new\ a:D_n;(P \mid\mid A[g_a/x])$$

$$decrypt\ \{b\}_a\ as\ \{x\}_a;P \rightarrow P[b/x] \qquad\qquad decrypt\ \{b\}_a\ as\{x\}_{g_a};A \rightarrow_{(g_a,0)} A[b/x]$$

$$decrypt\ \{b\}_a\ as\ \{x\}_{g_a};P \rightarrow_{(a)} (g_a)P[b/x] \qquad\qquad decrypt\ \{b\}_{g_a}\ as\ \{x\}_a;P \rightarrow_{(a)} (g_a)P[b/x]$$

For unguarded $C[_]$ if $P \rightarrow_\alpha P'$ then $C[P] \rightarrow_\alpha C[P']$

Figure 3: The Semantics of the pi-g calculus

correct confirmation of g_a if the attacker did not already know the encrypted value b. However, the truth of this assumption will depend on the exact implementation of encryption. So we err on the side of caution, and allow the attacker to confirm their guess by decryption alone.

As with the confirmation of a guess by an output, this may not be a definitive confirmation. It is possible that the process which generated the trace received the message $\{m\}_k$ and used it to confirm g_k, whereas, in another run of the same protocol the attack may receive a message encrypted with a different key. As above, we point out that the protocol should not rely on the scheduler for security. In Section 4 we give a full correctness argument. We show that the existence of even just one states that allows the attacker to confirm their guess is enough to give the non-negligible chance of failure needed to show that the protocol is computationally insecure.

The addition of encryption completes the syntax of our calculus:

Network	$::=$	$P \mid\mid A$
Processes P,Q,A	$::=$	0
& Attackers	$\mid$	$send\ a\langle\vec{b}\rangle$
	$\mid$	$rec\ a(\vec{x});P$
	$\mid$	$!P$
	$\mid$	$(\ P \mid Q\ )$
	$\mid$	$[a=b];P$
	$\mid$	$new\ a:D_n;P$
	$\mid$	$guess\ x:D_n;P$
	$\mid$	$decrypt\ a\ as\ \{x\}_k;P$

In general only attackers will use the *guess* operation and only processes will use secret names. The full semantics of the calculus is given in Figure 3. As the labels on a trace signal which bound names have been guess we can not later change these names. Therefore we not not allow alpha-conversion of names that have already been guess. If alpha-conversion is necessary then it must take place before the execution of the guess action. With this exception our structural equivalence rules are standard, and are giving in Figure 4.

4 The Correctness of the pi-g Calculus

This section gives a computational base to the calculus. We allow domain sizes to be parameterized on a security parameter and then show that if there is a successfully, finite attack in the pi-g calculus with less than exponential cost in the security parameter, then the process is unsafe in the computational setting. In particular, it can be defeated by a brute force guessing attack. This result justifies the design of the cost function given in Subsection 3.4, and in particular the reduction in the cost of an attack due to the confirmation of guesses.

The counterpart to this theorem, that the lack of a sub-exponential cost attack implies safety in the computational model, is harder to prove because the Turing machine attacker may be able to carry out attacks outside the model of the calculus. We could prove this safety theorem by way of the computational correctness

$$
\begin{array}{rclrclr}
P \mid Q & \equiv & Q \mid P & (P \mid Q) \mid R & \equiv & P \mid (Q \mid R) & \\
P \mid 0 & \equiv & P & rec\ a(\vec{b});P & \equiv & rec\ a(\vec{c});P[\vec{c}/\vec{b}] & \vec{c} \cap fn(P) = \{\} \\
!P & \equiv & P \mid !P & new\ a : D_n;P & \equiv & new\ b : D_n;P[b/a] & \{b, g_a\} \cup fn(P) = \{\} \\
new\ a : D_n;0 & \equiv & 0 & new\ a : D_n;P \mid Q & \equiv & new\ a : D_n;(P \mid Q) & \{a, g_a\} \cap fn(Q) = \{\} \\
 & & & new\ a : D_n;new\ b : D_n P & \equiv & new\ b : D_n;new\ a : D_n;P &
\end{array}
$$

Figure 4: The Structural Equivalence Rules

of the spi-calculus, which would be a useful result in itself. However, here we are only interested in showing that the guessing extensions to the spi-calculus are correct. So, in much the same way as computational security proofs often prove the safety of a system by relating it to another system that is believed to be safe, we show that the safety of the spi-calculus is enough to prove the safety of the pi-g calculus, and hence our extensions will introduce no new errors.

4.1 Relating the spi-calculus and the Computational Model

In this section, we formalise the relation between the spi-calculus and the computational model that we will use as a base for our correctness result.

We wish to pit a Turing Machine attacker against a protocol written in a process calculus. We do this by modelling the attacker as a Turing machine with an oracle that can run the process being attacked any number of times, for this we write A^P or $A^{P(\vec{a})}$ for the process P parameterise on the names $\vec{a}$.

The oracle can compute reductions of the calculus process P using the semantic rules for the calculus. The Turing machine A interacts with the oracle by means of two tapes, one for output, onto which the oracle will write outputs visible to the attacker and another tape for input, from which the oracle can read messages from the attacker and insert them in to the appropriate part of the process. Names are implemented as random bit strings that are at least as long as the security parameter, hence making them hard to guess in sub-exponential time. The process P would have no other access to the security parameter, unlike the attacker. To start a new run of the process the attacker writes a special symbol onto the tape along with a bit string. Appending this bit string then distinguishes the bit string names of this new concurrent run. Mitchell et al. describe a simple implementation of the pi-calculus [MRST01] where they aim to show the polynomial time reductions of a sub-set of the pi-calculus, whereas we do not care about the run time of the protocol, only the attacker. A fair scheduler is used to handle concurrent process, so if a process in a finite system can perform a reduction then, in the computational setting, there is a non-zero probability that it does perform that action.

We require that the encryption function used in the computational model is repetition concealing, whichkey concealing and message-length concealing. This extremely strong kind of encryption is termed type-0 security by Abardi and Rogaway [AR00] and is defined as making the following criterion negligible for all PRTMs.

$$
\begin{aligned}
Adv_Enc(n) \ &= \ Pr[k,k' \xleftarrow{r} Keys_n : A^{E_k(-),E_{k'}(-)}(n) = 1] \\
&- \ Pr[k \xleftarrow{r} Keys_n : A^{E_k(0),E_k(0)}(n) = 1]
\end{aligned}
$$

Encrypting a key with itself can be problematic, even when done indirectly [GM84]. So, we deviate slighting from the original spi-calculus and automatically consider process which do this unsafe.

In later work it might be interesting to consider an extension of our current system that would allow the attacker to easily guess a key from a message containing and encrypted by that key, or even a series of messages that indirectly encrypt one key with itself.

Definition 4.1 *$A^{(P_n)_{spi}}$ is the Turing machine A which has access to an oracle that behaves in a similar manner to the spi-calculus process P, as outlined above, in particular all bound names in P are mapped to bit strings at least as long as the security parameter n and A knows all the free names of P.*

It should be noted that the behaviour of the encoding of P_{spi} will not exactly match the behaviour of the spi-calculus process, because the attacking Turing machine A can find a bound name in exponential time. Instead, we match safety in the spi-calculus with safety against a polynomial time attacker in the computational setting.

To formalise exactly what we mean by a successful attack we consider the set of spi-calculus processes that are parameterised on a given constant $P(c)$. In the

spi-calculus, a process is considered safe when the process $P(c)$ is testing equivalenct to the process $P(a)$ for any name a. To relate this definition of safe to the computational model we reformalise this to say that a process $P(c)$ is safe if and only if there does not exist a process $A(c)$ such that $P(c) \mid\mid A(c)$ performs an output on c and $P(a) \mid\mid A(c)$ does not perform an output on c.

In the computational setting the attacker is given access to an oracle with a secret bit string, and the attacker is also given another, possibly different bit string. The attacker will answer "1" if it believes the value it was given is the same as the secret and "0" otherwise. The difference between these probabilities, of the attacker getting it right and getting it wrong, is the attackers advantage:

$$Adv(n) = Pr[s \stackrel{r}{\leftarrow} D_n \; : \; A^{P_n(s)_{spi}}(n, fn(P), s) = 1] \\ - Pr[s,t \stackrel{r}{\leftarrow} D_n \; : \; A^{P_n(s)_{spi}}(n, fn(P), t) = 1]$$

The attacker also has access to the free, public names of the process. We slightly abuse our notation here by using letters to represent names in the calculi processes and the bit strings that represent those names in the computational setting.

We say that the process P is safe in the computational setting if the advantage function $Adv(n)$ is negligible for all probabilistic, polynomial time Turing machines (PRTMs) A, otherwise we say it is unsafe. As mentioned above, we are interested in finding attacks on protocols that work in any reasonable situation, therefore we require the attacker A to work with non-negligible probability for any scheduler than assigns a non-zero, constant probability to any possible action.

Processes in the pi-g calculus can be treated in the same way. Except, when mapping these processes to the their computational equivalents, pi-g calculus names are mapped to random bit-strings with the same length as the size of their domains.

Definition 4.2 $A^{(P_n)_{pig}}$ *is the Turing machine A which has access to an oracle that behaves in a similar manner to the pi-g calculus process P_n, with the same mapping as that for spi-calculus processes except bound names are now mapped to bit strings of the same length as their domain size.*

4.2 Unsafe in the pi-g Calculus Implies Unsafe in the Computational Model

Now that we have a model in which a protocol can be attacked, we need to be sure that a successful attack in the model implies the possibility of a successful attack in an implementation of the protocol. We show this by proving that, if there exists an attack on a protocol in the pi-g calculus then the computational equivalent system can be broken by a polynomial time Turing machine.

Theorem 1 *Given a process with a secret value P(s), if there exists a sub-exponential cost, finite attack on the process in the pi-g calculus then there also exists a probabilistic, polynomial Turing machines that makes the advantage function pi-g $Adv(n)$ non-negligibly.*

$$pi\text{-}g\; Adv(n) = Pr[s \stackrel{r}{\leftarrow} D_n \; : \; A^{P_n(s)_{pig}(n,s)} = 1] \\ - Pr[s,t \stackrel{r}{\leftarrow} D_n \; : \; A^{P_n(s)_{pig}}(n,t) = 1]$$

PROOF: (Sketch)

Assume the existence of a sub-exponential cost attack in the pi-g calculus; we sketch the construction of a polynomial time Turing machine that will find the secret value with small but non-negligible probability.

The attack in the pi-g calculus must consist of a finite number of actions, however some of these actions are guesses, therefore we cannot map these finite actions directly to a finite Turing machines. Instead we show how each guess action can be removed in polynomial time with a non-negligible chance of success.

The attacking Turing machine will intercept the same outputs and send the same inputs to the oracle as attacking pi-g calculus process sent to the original process, i.e. if the pi-g calculus attacker did an output of the name a over the channel b followed by an input on the channel c the Turing machine attacker will send the bit string that represents a over the channel b and then receive another bit string from c.

The trace generated in the pi-g calculus represents just one possible reduction of the process. For instance, it is possible that the attacker will output a over the channel b and then the process non-deterministically choices some other path and does not perform an output on c. However the output c was found in a finite number of steps, and we are assuming a fair scheduler, so there is non-negligible probability that the protocol will behave in the same way as it did in the pi-g calculus trace.

This leaves us with the guess commands. When a pi-g calculus process guesses a value from domain of size n, the attacking Turing Machine will make n copies of itself onto n different tapes. Each of these copies then reruns the protocol to get to the same point in the attack. These reruns can be done in polynomial time, in the security parameter, and with a non-negligible chance of successes.

Each subsequent guess multiples the number of running machines, so if we guess values of size n then

m and then o, with will have $n \times m \times o$ machines running in parallel. When one of the Turing machines sees an action that verifies a guess in the pi-g calculus attack, that machine assumes that its guess is correct and halts all the machines that corresponded to other guesses.

As before, due to the non-determinism of the protocol, just because an action verifies a guess in one particular trace of a protocol, does not mean that the guess is verified in all possible traces. But, also as before, assuming a fair scheduler, there is a non-negligible probability that the guess has been correctly confirmed.

So, we have a parallel Turing machine that mimics the attack found in the pi-g calculus and takes polynomial time multiplied by the cost of the attack in the pi-g calculus. Hence, if the cost for the attack is sub-exponential, we have the Turing machines required by the computational criterion and the computation process is unsafe.

□

4.3 Safe in the pi-g Calculus Implies Safe in the Computational Model

A process, without any guesses, such as a protocol definition, can be mapped from the pi-g calculus to the spi-calculus by removing the domains from the new name operator. We can relate zero cost attacks in the pi-g calculus to attacks in the spi-calculus directly.

Proposition 4.1 *If a process in the pi-g calculus does not yield to successful zero cost attack, then the equivalent process in the spi-calculus is safe.*

PROOF:

If there is a successful attack on a process in the spi-calculus, then the similar attack will also work in the pi-g calculus. As the spi-calculus attack does not use any guesses so the cost of the attack in the pi-g calculus will be zero. Therefore the absence of a zero cost attack in the pi-g calculus implies that there cannot be an attack in the spi-calculus. □

This proposition provides a basic result, however there is a much more subtle questions that we must ask about our new calculus.

By extending and changing the spi-calculus model, to make the pi-g calculus, there is a possibility that we have allowed a new class of attacks that are outside the model of the pi-g calculus but are accounted for by the spi-calculus. This is especially possible with the switch from representing values as bit strings at least as long as the security parameter, as we do in the spi-calculus encoding, to representing values as possibly constant length strings, as we may in the pi-g calculus.

Theorem 2 *The pi-g calculus is a safe extension of the spi-calculus: If, for a spi-calculus process, the advantage function $Adv(n)$ is negligible, for all PRTMs only when there exist a successful spi-calculus attacker then, if a pi-g process P does not admit a sub-exponential cost attack then the advantage function pi-g $Adv(n)$ is negligible for all PRTMs.*

PROOF: (Sketch)

Let P_g be a pi-g process for which there are no sub-exponential attack. Assume, for contradiction, that there exists an PRTM A such that *pi-g* $Adv(n)$ is non-negligible and that no such attacker exists for $Adv(n)$ i.e. the attacker can defeat the Turing encoding of the pi-g calculus process, with its shorter length names, but cannot defeat the Turing encoding of the spi-calculus process. The existence of such a machine would mean that in extending the spi-calculus we had added a security hole that A could exploit.

As the cost of the attack is sub-exponential we know that the attacking process only guesses a finite number of sub-exponential names. So, we can write the process P_g as *new* $a_1 : D_{n_1}, .., a_i : D_{n_i}; Q$ where a_1 to a_i are the names guessed by the attacker. We use structural equivalence to unwind any replications that generated the guessed names. Now, back in the pi-g calculus, we consider the attacker that first guesses these names and then tries to find an attack on Q.

As the PRTM A breaks P_g, we can construct another machine, A_2 that breaks Q, by giving the true values of a_1 to a_i to A. However the part of Q involved in the attack can be considered as a spi-calculus process, and as we are assuming that the computational encoding of the spi-calculus is correct, there must exists a spi-calculus attacker that defeats Q, say R. Then the pi-g calculus process *guess* $a_1 : D_{n_1}, .., a_i : D_{n_i}; R$ defeats P_g with sub-exponential cost, giving us our contradiction.

□

5 Conclusions

We have presented an extension of the pi-calculus that can model simple guessing attacks. The new name operator in the pi-calculus is equated with random sampling in the computational model with the result that a new name can be guessed. We only allow a correct guess to originate

from a *guess* operator, this simplifies the cost analysis of attacks. In order to show correctness we related our work to the computational model. We assumed a translation of the spi-calculus into Turing machines.

Our first theorem proved that if the cost of an attack in our calculus is less than exponential then the process is unsafe because we can construct a PRTM attacker that defeats the process with non-negligible probability. Key to this theorem was that finding a confirmation of a guess in a single run of the pi-g calculus implies that a there is a non-negligible chance that the attacker also finds that confirmation in any future run.

Our other main result showed that if there are no sub-exponential attacks on a process then it is safe from guessing attacks. We prove that, when mapped to a computational setting, a pi-g calculus process that has no sub-exponential cost attack, is safe if and only if the equivalent process is safe in the spi-calculus.

Further work will include proving the computational correctness of the spi-calculus. This would follow the same lines as previous work on the computational correctness of the Dolev-Yao model.

An interesting extension to the calculus would be to let the pi-g calculus attacker make use of the security parameter. This might allow more interesting types of attackers and would require a modification of the cost function to take account of this new type of work done in time proportional to the security parameter.

We have also developed a labelled transition system semantics for the calculus. The reduction semantics was presented in this paper because we believe it is easier to understand. The LTS semantics could allow us to use bi-simulation to find more expressive equivalences for the calculus. Given the cost of the actions, the LTS semantics could also allow a metric on processes that would assign a value to any pair of processes reflecting how different the processes seem to a guessing attacker, perhaps along the lines of [DCPP05]. Our current semantics does not deal with some of the subtler points of confirmation. For instance, it would be possible for an attacker to receive a false confirmation if they incorrectly guessed one value but unknowingly guessed the true value of some other name. A more complicated confirmation rule might deal with such circumstances.

We are keen to apply the pi-g calculus to a real, large-scale, multi-user protocol in order to look for weak points. A bankcard system could be one possible example, as could a key distribution protocol. We are particularly interested in using this calculus to distinguish between secure systems where the leaking, or guessing, of a small number of values leads to the entire system being compromised and secure systems in which all secret values must be guessed in order to compromise the whole system. An automated checking system would make the analysis of large protocols in the pi-g calculus much more practical. An implementation of the calculus in Prolog might be a useful start, whereas the extension of an existing model checker would make a more usable tool.

References

[AG97] Martín Abadi and Andrew D. Gordon. A calculus for cryptographic protocols: The spi calculus. In *Fourth ACM Conference on Computer and Communications Security*, pages 36–47. ACM Press, 1997.

[AR00] Martín Abadi and Phillip Rogaway. Reconciling two views of cryptography (the computational soundness of formal encryption). In *IFIP International Conference on Theoretical Computer Science (IFIP TCS2000)*, Sendai, Japan, 2000. Springer-Verlag, Berlin Germany.

[Bac04] Michael Backes. A cryptographically sound Dolev-Yao style security proof of the Otway-Rees protocol. In *Proceedings of the 9th European Symposium on Research in Computer Security (ESORICS)*, 2004.

[BAN96] Michael Burrows, Martín Abadi, and Roger Needham. A logic of authentication, from proceedings of the Royal Society, volume 426, number 1871, 1989. In *William Stallings, Practical Cryptography for Data Internetworks, IEEE Computer Society Press*. 1996.

[BKR94] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of cipher block chaining. *Lecture Notes in Computer Science*, 839:341–358, 1994.

[BKR00] Mihir Bellare, Joe Kilian, and Phillip Rogaway. The security of the cipher block chaining message authentication code. *Journal of Computer and System Sciences*, 61(3):362–399, 2000.

[BM84] Manuel Blum and Silvio Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM Journal on Computing*, 13:850–864, 1984.

[BP04a] Michael Backes and Birgit Pfitzmann. A cryptographically sound security proof of the needham-schroeder-lowe public-key protocol. *Journal on Selected Areas in Communications*, 22(10), 2004.

[BP04b] Michael Backes and Birgit Pfitzmann. Symmetric encryption in a simulatable dolev-yao style cryptographic library. In *Computer Security Foundations Workshop*, 2004.

[BPW03] M. Backes, B. Pfitzmann, and M. Waidner. A composable cryptographic library with nested operations. In *10th ACM Conference on Computer and Communications Security (CCS)*, 2003.

[CDE03] R. Corin, J. M. Doumen, and S. Etalle. Analysing password protocol security against off-line dictionary attacks. In *Workshop on Security Issues with Petri Nets and other Computational Models (WISP)*, 2003.

[Cer01] Iliano Cervesato. A specification language for crypto-protocols based on multiset rewriting, dependent types and subsorting. In *Workshop on Specification, Analysis and Validation for Emerging Technologies*, pages 1–22, 2001.

[CMAFE03] R. Corin, S. Malladi, J. Alves-Foss, and S. Etalle. Guess what? Here is a new tool that finds some new guessing attacks. In *Workshop on Issues in the Theory of Security (WITS)*, 2003.

[DCPP05] Yuxin Deng, Tom Chothia, Catuscia Palamidessi, and Jun Pang. Metrics for action-labelled quantitative transition systems. In *Workshop on Quantitative Aspects of Programming Languages (QAPL)*, 2005.

[DJ04] S. Delauune and F. Jacquemard. A theory of guessing attacks and its complexity. Technical report, ENS de Cachan, 2004.

[DY83] D. Dolev and A.C. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, 1983.

[FA03] Cédric Fournet and Martín Abadi. Hiding names: Private authentication in the applied pi calculus. In *Proceedings of the International Symposium on Software Security (ISSS'02)*, volume 2906 of *LNCS*, pages 317–338, 2003.

[GGM86] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM*, 33(4):210–217, 1986.

[GLNS93] L. Gong, M. A. Lomas, R. M. Needham, and J. H. Saltzer. Protecting poorly chosen secrets from guessing attacks. *IEEE Journal on Selected Areas in Communications*, 11(5):648–656, 1993.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, 28(2):270–299, 1984.

[GMR88] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof-systems. *SIAM Jounral on Computing*, 17:281–308, 1988.

[Her04] Jonathan Herzog. *Computational Soundness for Standard Assumptions of Formal Cryptography*. PhD thesis, Massachusetts Institute of Technology, May 2004.

[Hüt02] H. Hüttel. Deciding framed bisimilarity. In Antonn and Mayr, editors, *4th International Workshop on Verification of Infinite-State Systems, Infinity'02 ENTCS*, volume 68(6), 2002.

[JML05] Romain Janvier, Laurent Mazare, and Yassine Lakhnechs. Completing the picture: Soundness of formal encryption in the presence of active adversaries. In *Proceedings of the European Symposium on Programming*, 2005.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. *Lecture Notes in Computer Science*, 1109:104–113, 1996.

[Lau04] Peeter Laud. Symmetric encryption in automatic analyses for confidentiality against active adversaries. In *Proceedings of 2004 IEEE Symposium on Security and Privacy*, pages 71–85, 2004.

[Low96] Gavin Lowe. Breaking and fixing the Needham-Schroeder public-key protocol using FDR. In *Tools and Algorithms for*

the Construction and Analysis of Systems (TACAS), volume 1055, pages 147–166. Springer-Verlag, Berlin Germany, 1996.

[Low02] Gavin Lowe. Analysing protocols subject to guessing attacks. In *Proceedings of the Workshop on Issues in the Theory of Security (WITS)*, 2002.

[Mil91] Robin Milner. The polyadic pi-calculus - a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for the Foundations of Computer Science, 1991.

[MMS03] P. Mateus, J.C. Mitchell, and A. Scedroc. Composition of cryptographic protocols in a probabilistic polynomial-time process calculus. In *CONCUR 2003 - Concurrency Theory, 14-th International Conference, Marseille, France, September*, 2003.

[MRST01] J. Mitchell, A. Ramanathan, A. Scedrov, and V. Teague. A probabilistic polynomial-time calculus for analysis of cryptographic protocols (preliminary report). In S. Brookes and M. Mislove, editors, *17-th Annual Conference on the Mathematical Foundations of Programming Semantics*, volume 45, Aarhus, Denmark, 2001.

[MW04] D. Micciancio and B. Warinschi. Soundness of formal encryption in the presence of active adversaries. In *Proceedings of the Theory of Cryptography Conference*, pages 133–155. Springer, 2004.

[RMST04] A. Ramanathan, J. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *Foundations of Software Science and Computation Structures, 7-th International Conference, FOSSACS*, 2004.

[THG99] J. Thayer, J. Herzog, and J. Guttman. Strand spaces: Proving security protocols correct. *Journal of Computer Security*, 7(2/3):191–230, 1999.

[War03] B. Warinschi. A computational analysis of the Needham-Schroder(-Lowe) protocol. In *Proceedings of the 16th Computer Secuirty Foundations Workshop*, pages 248–262, 2003.

[ZD04] Roberto Zunino and Pierpaolo Degano. A note on the perfect encryption assumption in a process calculus. In *Foundations of Software Science and Computation Structures: 7th International Conference, FOSSACS 2004*, volume 2987/2004 of *Lecture Notes in Computer Science*, pages 514–528. Springer-Verlag, 2004.