Basic Research in Computer Science

BRICS RS-00-6 Damgård & Nielsen: Improved Non-Committing Encryption Schemes

Improved Non-Committing Encryption Schemes based on a General Complexity Assumption

Ivan B. Damgård
Jesper Buus Nielsen

BRICS Report Series **RS-00-6**

ISSN 0909-0878 **March 2000**

**See back inner page for a list of recent BRICS Report Series publications.
Copies may be obtained by contacting:**

**BRICS
Department of Computer Science
University of Aarhus
Ny Munkegade, building 540
DK–8000 Aarhus C
Denmark
Telephone: +45 8942 3360
Telefax: +45 8942 3255
Internet: BRICS@brics.dk**

BRICS publications are in general accessible through the World Wide Web and anonymous FTP through these URLs:

`http://www.brics.dk`
`ftp://ftp.brics.dk`
This document in subdirectory `RS/00/6/`

Improved Non-Committing Encryption Schemes based on a General Complexity Assumption

Ivan B. Damgård Jesper Buus Nielsen

March, 2000

Abstract

Non-committing encryption enables the construction of multiparty computation protocols secure against an *adaptive* adversary in the computational setting where private channels between players are not assumed. While any non-committing encryption scheme must be secure in the ordinary semantic sense, the converse is not necessarily true. We propose a construction of non-committing encryption that can be based on any public key system which is secure in the ordinary sense and which has an extra property we call *simulatability*. The construction contains an earlier proposed scheme by Beaver based on the Diffie-Hellman problem as a special case, and we propose another implementation based on RSA. In a more general setting, our construction can be based on any collection of trapdoor one-way permutations with a certain simulatability property. This offers a considerable efficiency improvement over the first non-committing encryption scheme proposed by Canetti et al. Finally, at some loss of efficiency, our scheme can be based on general collections of trapdoor one-way permutations without the simulatability assumption, and without the common domain assumption of Canetti et al.

1 Introduction

The problem of multiparty computation dates back to the papers by Yao [13] and Goldreich et al. [8]. What was proved there was basically that a collection of n players can efficiently compute the value of an n-input function, such that everyone learns the correct result, but no other new information. More precisely, these protocols can be proved secure against a polynomial time bounded adversary who can *corrupt* a set of less than $n/2$ players initially, and then make them behave as he likes. Even so, the adversary should not be able to prevent the correct result from being computed and should learn nothing more than the result and the inputs of corrupted players. Because the set of corrupted players is fixed from the start, such an adversary is called *static* or non-adaptive.

There are several different proposals on how to define formally the security of such protocols [12, 2, 5], but common to them all is the idea that security means that the adversary's view can be *simulated* efficiently by a machine that has access to only those data that the adversary is *entitled* to know. Proving correctness of a simulation in the case of [8] requires a complexity assumption, such as existence of trapdoor one-way permutations. This because the protocol takes place on an open network where the adversary may potentially see every (encrypted) message exchanged between players.

Later, *unconditionally* secure MPC protocols were proposed by Ben-Or et al. and Chaum et al.[3, 6], in the model where *private* channels are assumed between every pair of players. These protocols are in fact secure, even if the adversary is *adaptive*, i.e. can choose dynamically throughout the protocol who to corrupt, as long as the total number of corruptions is not too large. It is widely accepted that adaptive adversaries model realistic attacks much better than static ones. Thus it is natural to ask whether adaptive security can also be obtained in the computational setting?

The protocol from [8] is not known to be adaptively secure. The original simulation based security proof for [8] fails completely against an adaptive adversary. However, in [4], Canetti et al. introduce a new concept called *non-committing encryption* and observe that if one replaces messages on the secure channels used in [3, 6] by non-committing encryptions sent on an open network, one obtains adaptively secure MPC in the computational setting. They also showed how to implement non-committing encryption based on so called common-domain trapdoor one-way permutations. The special property of non-committing encryption (which ordinary public-key encryption lacks) is the following: although a normal ciphertext determines a clear-text uniquely, encrypted communication can nevertheless be simulated with an indistinguishable distribution such that the simulator can later "open" a ciphertext to reveal any clear-text it desires. In an MPC setting, this is what allows to simulate the adversary's view before *and after* a player is corrupted.

Subsequently, Beaver [2] has proposed a scheme based on the Decisional Diffie-Hellman assumption (DDH) that is much simpler than the scheme from [4].

2 Our Results

In this paper, we propose a general way to build non-committing encryption from ordinary semantically secure public-key encryptions schemes. Our method offers a major efficiency improvement over [4] if the scheme we start from has an extra property we call *simulatability*. Roughly speaking, a public-key scheme is simulatable if, in addition to the normal key generation procedure, there is an algorithm to generate a public key, without getting to know the corresponding secret key. Moreover, it must be possible to sample efficiently a random ciphertext without getting to know the corresponding clear-text (we give precise definitions later in the paper).

The idea that it could be useful to generate a public key without knowing the secret key is not new. It seems to date back to De Santis et al.[7] where it was used in another context. The idea also appears in [4], but was only used there to improve the key generation procedure is some special cases. Here, we show the following

- From any semantically secure and simulatable public-key system, one can construct a non-committing encryption scheme.
- The scheme requires 3 messages to communicate k encrypted bits, where k is the security parameter. The total amount of communication is $O(k)$ public keys, $O(k)$ encryptions of a k bit clear-text (in the original scheme), and k bits.
- Only the final k bits of communication depend on the actual message to be sent, and hence nearly all the work needed can be done in a preprocessing phase.

Our scheme is inspired by the one of Beaver[2], and indeed if we use as basis of our construction the encryption scheme that follows naturally from the Decisional Diffie-Hellman (DDH) assumption, we get something that is essentially equivalent to [2]. We propose an alternative implementation based on the RSA assumption. This scheme is almost as efficient as the DDH scheme (it is slower by at most a $\log^2 k$ factor, all other things being equal).

We then look at general families of trapdoor one-way permutations. Call such a family simulatable if one can efficiently generate a permutation in the family without getting to know the trapdoor and the domain can be sampled in an invertible manner[1] We show that such a simulatable family implies immediately a simulatable public-key system with no further assumptions. The non-committing encryption scheme we obtain from this requires per encrypted bit communicated that we send $O(1)$ descriptions of a permutation in the family and $O(k)$ bits (where the hidden constant only has to be larger than 2, and where all bits except one can be sent in a preprocessing phase). With the same assumption, the scheme from [4] requires $\Omega(1)$ permutation descriptions and $\Omega(k^2)$ bits. Moreover, the $\Omega(k^2)$ bits depend on the message communicated and so cannot be pushed into a preprocessing phase.

Finally, we observe that, at a loss of efficiency, we can implement non-committing encryption from any family of trapdoor one-way permutations, assuming only invertible sampling, i.e., without assuming full simulatability or the common domain assumption of [4]. For this, we use as subrutine a key generation protocol shown in [4]. We identify a bug in that protocol which caused it to be insecure as stated originally, and we suggest a modification under which it is secure. The protocol is based on oblivious transfer and establishes a situation where a player knows the trapdoor for one out of two public trapdoor permutations. Our scheme can then start from

[1]This is a technical condition which we discuss in more detail later. All known examples of trapdoor permutations have invertible sampling.

this situation and work with no extra assumptions. In this case the efficiency improvement over [4] is not so significant because the cost of the key generation will dominate the rest. The scheme retains the preprocessing capability, however.

We note that the key generation of [4] in fact needs invertible sampling in any case, and thus our assumption of existence of one-way trapdoor permutations with invertible sampling is the weakest known assumption sufficient for non-committing encryption.

3 Simulatable Public-Key Systems

Throughout the paper we will use the following notation. For a probabilistic algorithm $\mathcal{A}$ we will by $\mathcal{R}_{\mathcal{A}}$ denote a sufficiently large set $\boldsymbol{B}^l = \{0,1\}^l$ from which the random bits for $\mathcal{A}$ are drawn. We let $r \leftarrow \mathcal{R}_{\mathcal{A}}$ denote a r drawn uniformly random from $\mathcal{R}_{\mathcal{A}}$, let $a \leftarrow \mathcal{A}(x,r)$ denote the result a of evaluating $\mathcal{A}$ on input x using random bits r, and denote by $a \leftarrow \mathcal{A}(x)$ a value a drawn from the random variable $\mathcal{A}(x)$ describing $\mathcal{A}(x,r)$ when r is uniform over $\mathcal{R}_{\mathcal{A}}$.

We now want to define a public key encryption scheme where one can generate a public key without getting to know the matching secret key. So in addition to the normal key generation algorithm $\mathcal{K}$ that outputs a public and secret key (P,S), we assume that there is another algorithm which we call the oblivious key generator $\tilde{\mathcal{K}}$ which outputs only a public key P with a distribution similar to public keys produced by $\mathcal{K}$. However, this condition is not sufficient to capture what we want. $\tilde{\mathcal{K}}$ could satisfy it by just running the same algorithm as $\mathcal{K}$ but output only P. We therefore also ask that based only on a properly generated key P, there is an efficient algorithm $\tilde{\mathcal{K}}^{-1}$ that comes up with a set of random choices r' for $\tilde{\mathcal{K}}$ such that $P = \tilde{\mathcal{K}}(r')$ and P, r' cannot be distinguished from a normal set of random choices and resulting output from $\tilde{\mathcal{K}}$. This ensures that whatever side information you get from producing P using $\tilde{\mathcal{K}}$, you could also compute efficiently from only P itself. In a similar way we can define what it means to produce a random ciphertext with no knowledge of the plaintext. Formalising this we get:

Definition 1 (Simulatable public-key system) *Let $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{M})$ be a public-key system with key-generation algorithm $\mathcal{K}$, encryption algorithm $\mathcal{E}$, decryption algorithm $\mathcal{D}$, message-space generator $\mathcal{M}$, and security parameter k (1^k is implicitly given as input to all algorithms in the following). We say that $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{M})$ is a* simulatable public-key system *if the following probabilistic polynomial time algorithms exist: the* oblivious public-key generator $\tilde{\mathcal{K}}$*; the* key faking algorithm $\tilde{\mathcal{K}}^{-1}$*; the* oblivious cipher-text generator $\mathcal{C}$*; and the* cipher-text faking algorithm $\mathcal{C}^{-1}$*. And if furthermore the following holds:*

Oblivious public-key generation *For $r \leftarrow \mathcal{R}_{\mathcal{K}}$, $(P,S) \leftarrow \mathcal{K}(r)$, $r' \leftarrow \tilde{\mathcal{K}}^{-1}(P)$, and $\tilde{r} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}$, $\tilde{P} \leftarrow \tilde{\mathcal{K}}(\tilde{r})$ the distributions (r', P) and $(\tilde{r}, \tilde{P})$ are computationally indistinguishable.*

Oblivious cipher-text generation *For* $(P,S) \leftarrow \mathcal{K}$, $r_1 \leftarrow \mathcal{R}_{\mathcal{C}}$, $C_1 \leftarrow \mathcal{C}(P,r_1)$, *and* $M \leftarrow \mathcal{M}_P$, $r_2 \leftarrow \mathcal{R}_{\mathcal{E}}$, $C_2 \leftarrow \mathcal{E}_P(M,r_2)$, $r_2' \leftarrow \mathcal{C}^{-1}(C_2,P)$ *the random variables* (P,r_1,C_1) *and* (P,r_2',C_2) *are computationally indistinguishable.*

Semantic security *For* $r \leftarrow \mathcal{R}_{\mathcal{K}}$, $(P,S) \leftarrow \mathcal{K}(r)$, *and for* $i = 0,1$*:* $M_i \leftarrow \mathcal{M}_P$, $r_i \leftarrow \mathcal{R}_{\mathcal{E}}$, $C_i \leftarrow \mathcal{E}_P(M_i,r_i)$, *the random variables* (P,M_0,M_1,C_0) *and* (P,M_0,M_1,C_1) *are computationally indistinguishable.*

4 Non-Committing Encryption

Many proposals for the definition of secure multiparty computation has appeared in the literature presently culminating in the proposal of [5] which as the first definition allows for general security preserving modular composition of protocols in the computational setting. We will use this model of secure multiparty computation and sketch it here. A complete definition appears in [5].

The goal is for n parties to securely evaluate a function f taking an input from each party and outputting a private output to each party. In formulating security first an ideal evaluation is defined which captures the most we can expect of security. A real life execution of the protocol is then required to resemble this ideal evaluation.

Besides the n parties of the protocol three other entities participate in the ideal evaluation. Two probabilistic polynomial time algorithms, the adversary $\mathcal{S}$ and the environment $\mathcal{Z}$, and an oracle, called the trusted party $\mathcal{T}$, for computing f. Each party share a secure channel with $\mathcal{T}$. The evaluation proceeds as follows. First the adversary adaptively corrupts a number of parties and possible alters their inputs [2]. The parties then send their (possibly altered) inputs to the trusted party, which returns to each party its share of the output of f. This is done over secure channels and the adversary learns nothing. Then again the adversary adaptively corrupts a number of parties and learns their inputs and outputs. A corrupted party stays corrupted and the adversary therefore receives also the outputs of the parties corrupted in the first round of corruption. The uncorrupted parties then output the value received from the trusted party, the corrupted parties output $\perp$, and the adversary outputs some arbitrary value computed on the values learned through corruption. Before this process the environment hands some arbitrary value to $\mathcal{S}$. During the process the environment learns the identity of corrupted parties and after each corruption the environment passes a message to the adversary. After termination of the evaluation the environment receives the output values of the parties and the adversary. Then a post-execution corruption phase starts, were the environment might corrupt

[2] The definition also covers the cases where the adversary is passive (i.e., he just monitors the parties without changing anything), or is static, (i.e., the set of corrupted parties is chosen once and for all.) We concentrate here on the adaptive, active case

more parties - through interaction with $\mathcal{S}$. Finally the environment outputs some arbitrary value computed on the obtained information. [3]

In the real life execution the parties follows the protocol π. Prior to each round of communication the adversary $\mathcal{A}$ adaptively corrupts a number of parties and learns all their internal data and random values. The adversary then decides for each corrupted party, which values it should communicate in this round. Furthermore, we look here at the scenario without private channels, so the adversary sees all communication. After protocol termination the uncorrupted parties then output according to the protocol and corrupted parties output $\perp$. Again the adversary outputs some arbitrary value. The interaction with the environment $\mathcal{Z}$ is the same as in the ideal evaluation.

We say that a protocol π t-adaptively securely computes f if for any environment $\mathcal{Z}$ and any real life t-adaptive adversary $\mathcal{A}$ there exists an ideal model t-adaptive adversary $\mathcal{S}$ such that for any input value to the protocol the output of $\mathcal{Z}$ after a real life execution of π with adversary $\mathcal{A}$ in environment $\mathcal{Z}$ is distributed computationally indistinguishably from the output of $\mathcal{Z}$ after an ideal evaluation with the same input value and adversary $\mathcal{S}$ in the same environment.

The only known proof technique for proving existence of the ideal model adversary $\mathcal{S}$ is a constructive one known as a simulation argument. Given any real life adversary $\mathcal{A}$, which expects to attack a real life execution, $\mathcal{S}$ translates the corruption requests of $\mathcal{A}$ into corruption requests on an ideal evaluation. The ideal model adversary $\mathcal{S}$ then receives the input and possibly the output of the corrupted party. However, after the corruption $\mathcal{A}$ expects to see all internal data (according to the real life protocol) of the corrupted party. To meet this requirement $\mathcal{S}$ will internally run a *simulated* copy of the real life protocol.

Since the real life protocol takes place in an open network where the adversary may potentially see all messages sent, and since the adversary is adaptive, the following problem occurs when trying to simulate such a protocol: the simulator has to construct something that looks like the communication taking place between uncorrupted parties, since in real life, the adversary would be able to see this information. This communication will typically be encrypted, so the simulator may try to put encryptions of random values in stead of real messages (at this point, neither the adversary nor the simulator knows what an honest party would actually send). However, the sender or receiver may later be corrupted, and then the adversary gets to know all inputs of the corrupted party. This data was unknown when the simulator

[3] A protocol might be carried out as a sub-protocol of some enclosing protocol. The messages passed from the environment to the adversary, when a party is corrupted, models the data of the corrupted party in the enclosing protocol - data flow from the enclosing protocol to the sub-protocol. The post-corruption phase models the fact, that a party might get corrupted in the enclosing protocol after the sub-protocol has terminated, in which case an adversary will also learn the data of the terminated sub-protocol - data flow from the sub-protocol to the enclosing protocol. Introducing the environment as a generalisation of the auxiliary input to the adversary in prior models is one of the important contributions of [4]. This is exactly what allows for security preserving modular composition.

had to produce the simulated ciphertexts, so most likely those ciphertexts will not match the inputs, at least not if a ciphertext always determines a cleartext uniquely, as is the case for ordinary encryption schemes.

This problem may seem unsolvable: a ciphertext must determine the cleartext uniquely in order for decryption to be possible. Nevertheless, the notion of non-committing encryption [4] offers a solution to this problem. The idea is that one might be able to let the simulator generate fake ciphertexts with a distribution indistinguishable, but different from that of real ciphertexts. And that fake ciphertexts may be constructed to be non-committing, i.e., the simulator can later make them seem consistent with cleartexts of its choice.

To define non-committing encryption more precisely, we follow [4] and define what our goal is, rather than fixing an implementation strategy:

Definition 2 (Non-committing encryption[4]) *Let $n > 2$ and let $f(b, \Lambda, \Lambda, \ldots, \Lambda) = (\Lambda, b, \Lambda, \ldots, \Lambda)$ be the n-party function for communicating a bit from party 1 to party 2 (Λ denotes the empty string.) Let π be an n-party protocol. We say that π is a non-committing encryption scheme if it securely computes f in the presence of $(n-1)$-adaptive adversaries.*

We introduce and provide examples of protocols adhering to a stronger notion of non-committing encryption which is resilient against a full attack.

Definition 3 (Strong Non-committing encryption) *Let $f(b, \Lambda) = (\Lambda, b)$ be the two-party function for communicating a bit $b \in \boldsymbol{B}$. Let π be a two-party protocol. We say that π is a strong non-committing encryption scheme if it securely computes f in the presence of 2-adaptive adversaries.*

The definition of strong non-committing encryption can be extended to n parties in the line of definition 2. Note that strong non-committing encryption trivially implies strong n-party non-committing encryption. Whether the other implication holds is an open problem. It is certainly not trivial as more parties than the communicating ones might take active part in the protocol.

4.1 The Main Idea

The basic idea in the protocol is - like in all previous proposals - that we have our parties learn less information than is actually possible. This opens the possibility that a simulator can choose to learn full information and exploit this to its advantage. A simulatable encryption scheme $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{M})$ has two properties which allow parties to learn less than maximal information. First, we can pick a public key $P \leftarrow \tilde{\mathcal{K}}$ without learning anything about a corresponding private key. Second, we can pick an encryption $C \leftarrow \mathcal{C}_P$ without learning anything about the corresponding plaintext. Using these properties the parties S and R can - in an adaptively secure way

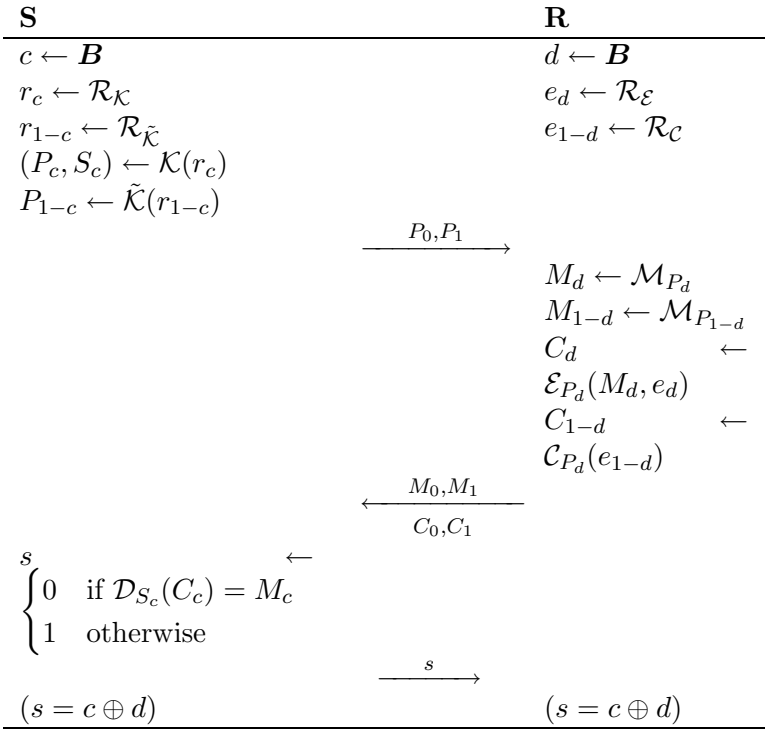

Figure 1: One attempt to establish a shared random bit.

- agree on a random, secret bit and then use this bit as a one-time pad. The main building block of the protocol, which we call an attempt, is sketched in Fig. 1.

First S chooses a bit c and generates a pair of public keys (P_0, P_1) such that he only knows the secret key corresponding to P_c. Then R chooses a bit d at random and produces two pairs of ciphertext/plaintext $(C_0, M_0), (C_1, M_1)$, such that only one pair is valid, i.e., C_d is an encryption of M_d under P_d, whereas the other pair is randomly chosen. It is now clear that S can determine, using the secret key he knows, whether $c = d$. The only possibility of error is that the pair (C_{1-d}, M_{1-d}) may be valid by coincidence. But if the messagespace has size superpolynomial in k (which we may assume without loss of generality), this happens with negligible probability. Finally, S sets $s = 0$ if and only if he found that $c = d$, and sends s to R. If $c = d$, the parties will use this secret bit to communicate message bit m securely as $m \oplus d$. If $c \neq d$, we say that the attempt has failed, and none of the bits c, d are used later [4]. We now do an initial analysis of the attempt protocol.

Let r_S and r_R be the random inputs of S resp. R. We write the values obtained by an attempt as

$$\text{ATTEMPT}(r_S, r_R) = (r_c, P_c, M_c, e_c, C_c), S_c, (r_{c-1}, P_{c-1}, M_{c-1}, e_{c-1}, C_{c-1}), (c, d, s).$$

[4]This may seem strange, since even in this case one party knows the choice of the other. However, for technical reasons that we return to later, the protocol would not be adaptively secure if all attempts were used for communication

Let ATTEMPT denote the random variable describing ATTEMPT(r_S, r_R) when r_S and r_R are chosen uniformly random. Let ATTEMPT$_i$ for $i = 0, 1$ denote the distribution of ATTEMPT under the condition that $s = i$. An attempt where $s = 0$ is called a successful attempt and an attempt where $s = 1$ is called a failed attempt.

proc SIMATTEMPT$_0$ $\equiv$
$s \leftarrow 0$
$r_i \leftarrow \mathcal{R}_{\mathcal{K}}, i = 0, 1$
$(P_i, S_i) \leftarrow \mathcal{K}(r_i), i = 0, 1$
$M_i \leftarrow \mathcal{M}, i = 0, 1$
$e_i \leftarrow \mathcal{R}_{\mathcal{E}}, i = 0, 1$
$C_i \leftarrow \mathcal{E}_{P_i}(M_i, e_i)$

proc SIMATTEMPT$_1$ $\equiv$
$s \leftarrow 1$
$c \leftarrow \boldsymbol{B}$
$r_c \leftarrow \mathcal{R}_{\mathcal{K}}$
$r_{1-c} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}$
$(P_c, S_c) \leftarrow \mathcal{K}(r_c)$
$P_{1-c} \leftarrow \tilde{\mathcal{K}}(r_{1-c})$
$d \leftarrow c - 1$
$M_i \leftarrow \mathcal{M}, i = 0, 1$
$e_d \leftarrow \mathcal{R}_{\mathcal{E}}$
$e_{1-d} \leftarrow \mathcal{R}_{\mathcal{C}}$
$C_d \leftarrow \mathcal{E}_{P_d}(M_d, e_d)$
$C_{1-d} \leftarrow \mathcal{C}_{P_d}(e_{1-d})$

Figure 2: Preprocessing values for the simulator, leaving the values of c and d open in the case $c = d$, i.e when $s = 0$.

For later use in the simulator and for illustration of the main idea we now show how we can produce a distribution computationally indistinguishable from that of ATTEMPT, but where (in case $c = d$) the value of the shared secret bit can later be changed. Let SIMATTEMPT$_i$ for $i = 0, 1$ be the distributions describing the values produced by the algorithms in Fig. 2. Let SIMATTEMPT be the distribution where an element is drawn from either SIMATTEMPT$_0$ or SIMATTEMPT$_1$ with probability $\frac{1}{2}$. The only difference compared to ATTEMPT is that in case of $SIMATTEMPT_0$, we choose to learn the corresponding private key of P_{1-c} and choose C_{1-d} as an encryption of M_{1-d}. However, to an outsider, who does not see S_{1-c} and e_{1-d}, this is not visible. We can therefore use the distribution SIMATTEMPT in a simulation in place of ATTEMPT to generate values of P_i, M_i, C_i and s for the communication. Furthermore, we can later "open" attempts where $c = d$ by fixing $c = d$ to any desired value - this is possible since S_{1-c} will never be given to the adversary, so he cannot see that the simulator chose both (M_d, C_d) and (M_{1-d}, C_{1-d}) as correct message/encryption pairs. He can only see that $D_{S_c}(C_d) = M_d$ as it should be. [5] For the purpose of patching successful attempts we define the function PATCH(S_0, b), which for an element S_0 drawn from SIMATTEMPT$_0$ and a bit $b \in \boldsymbol{B}$ produces values similar to those in ATTEMPT by computing c and d by $c \leftarrow b$, $d \leftarrow b$, and patching r_{1-c} and e_{1-c} by $r'_{1-c} \leftarrow \tilde{\mathcal{K}}^{-1}(P_{1-c})$, $e'_{1-c} \leftarrow \mathcal{C}^{-1}(C_{1-c}, P_{1-c})$.

Some notation: let PATCH $= (r_c, P_c, M_c, e_c, C_c), S_c, (r'_{1-c}, P_{1-c}, M_{1-c}, e'_{1-c}, C_{1-c}), (c, d, s)$ denote the random variable describing PATCH(S_0, b) when S_0 is drawn randomly from SIMATTEMPT$_0$ and b is drawn uniformly random from $\boldsymbol{B}$. Let PATCH^0 and PATCH^1 denote the distributions obtained by fixing $b = 0$, resp. $b = 1$. Finally, let ATTEMPT$_0^b$ denote the distribution of ATTEMPT$_0$ conditioned

[5] Consider instead the situation where $c \neq d$. The secret key S_c is always known by S. If this key becomes know to the adversary by corrupting S, he can check whether $\mathcal{D}_{S_c}(C_{1-d}) \neq M_{1-d}$, as it should be with high probability. The simulator can therefore not choose both (M_d, C_d) and (M_{1-d}, C_{1-d}) as correct message/encryption pairs when $c \neq d$ - (M_{1-d}, C_{1-d}) should be chosen randomly as in SIMATTEMPT$_1$. This implies that when simulating $s = 1$ the value of d is fixed from the round where (M_{1-d}, C_{1-d}) is communicated. This is why the d (or equivalently c) bit from failed attempts cannot be used for communicating a message bit - they cannot later be patched to new values consistent with the bit communicated in the ideal evaluation.

on $c = d = b$. We then have:

Lemma 1 *The distribution of* SIMATTEMPT_1 *is equal to the distribution of* ATTEMPT_1. *The distribution of* PATCH^b *is computationally indistinguishable from the distribution of* ATTEMPT_0^b *for* $b = 0, 1$ *in particular* PATCH *is computationally indistinguishable from* ATTEMPT_0

Proof: Obviously $\text{SIMATTEMPT}_1 = \text{ATTEMPT}_1$ by inspection of Figs. 1 and 2. For $s = 0$ let b denote the common value of c and d and observe that $\Pr[b = 0] = \Pr[b = 1] = \frac{1}{2}$ in both PATCH and ATTEMPT_0. It is therefore enough to show that the distributions of PATCH^b and ATTEMPT_0^b are computationally indistinguishable for $b = 0, 1$. For fixed b the variables c, d, and s are constants and has the same values in the two distributions, so we can exclude them from the analysis. Further more $(r_c, P_c, M_c, e_c, C_c), S_c$ can be seen to have the same distribution in the two distributions and is independent of $(r_{1-c}, P_{1-c}, M_{1-c}, e_{1-c}, C_{1-c})$, so all that remains is to show that these $(1-c)$-values are distributed computationally indistinguishable in ATTEMPT_0^b and PATCH^b. In ATTEMPT_0^b these values are distributed as

$$(\tilde{r} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}, \tilde{P} \leftarrow \tilde{\mathcal{K}}(\tilde{r}), M \leftarrow \mathcal{M}_{\tilde{P}}, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_{\tilde{P}}(e)) \tag{1}$$

and in PATCH^b they are distributed as

$$(r', P, M \leftarrow \mathcal{M}_P, e', C \leftarrow \mathcal{E}_P(M, e)) \tag{2}$$

where $r \leftarrow \mathcal{R}_{\mathcal{K}}$, $(P, S) \leftarrow \mathcal{K}(r)$, $r' \leftarrow \tilde{\mathcal{K}}^{-1}(P)$ and $e \leftarrow \mathcal{R}_{\mathcal{E}}$, $e' \leftarrow \mathcal{C}^{-1}(C.P)$. That these distributions are computationally indistinguishable follows by a hybrids argument, going from (1) to (2) using (in this order) the oblivious key property, the oblivious cipher-text property, and finally the semantic security property. For more details see lemma 3 in appendix C ■

4.2 The full protocol

The ability to patch a successful attempt to any value b of c and d provides enough that we can build a simulator for protocols that use the attempt building block 1 and which only uses bits obtained from successful attempts. To be sure of obtaining enough successful attempts one can either run attempts in sequence until enough successful attempt have occurred or run a number of independent attempts in parallel.

We will here analyse a protocol where a number of independent attempts are run in parallel to obtain $l(k)$ successful attempts for some polynomial $l(k) \in \Omega(k)$. Each attempt has probability $\frac{1}{2}$ of being successful, so it follows directly from the Markov inequality that $a(k) = 4l(k)$ attempts will give $l(k)$ successful ones except with probability $\exp(-\frac{l(k)}{2})$, which is certainly negligible in k. The goal of the protocol will be to evaluate the 2-party function $f(m, \Lambda) = (\Lambda, m)$, where $m \in \boldsymbol{B}^{l(k)}$ We do this by making $4l(k)$ attempts in parallel, letting b be the string of c-values from

the first $l(k)$ successful attempts, and finally communicate m by sending $f = m \oplus b$. In the negligible few cases, where we do not get $l(k)$ successful attempt we can do whatever, say we fill b with zeros. For a detailed description of the full parallel protocol see appendix B.

We proceed to generalise lemma 1 to the full setting. Let $\text{ATTEMPT}^l(m)$ be the random variable describing the distribution of values produced by a real-life execution of the full parallel protocol on input $m \in \boldsymbol{B}^{l(k)}$, i.e. $a(k)$ independent elements from ATTEMPT, the b and the f values. Let SIMATTEMPT^l be the values produced by drawing $a(k)$ independent elements from SIMATTEMPT and drawing $f \leftarrow \boldsymbol{B}^{l(k)}$ uniformly at random. Recall that the value of b is not specified yet since the values of c and d in SIMATTEMPT_0 are not fixed until patching. Let for $m \in \boldsymbol{B}^l$ and an element $S \in \text{SIMATTEMPT}^l$ the random variable $\text{PATCH}^l(S, m)$ be S extended with $b \leftarrow m \otimes f$, where further more the l first successful attempts in S are patched using b[6]. The remaining successful attempts are patched with uniformly random bits. Since f is uniformly random so are $b = f \oplus m$. Therefore all successful attempts are patched using uniformly random bits. This will allow us to use Lemma 1. Let $\text{PATCH}^l(m)$ be a random variable describing $\text{PATCH}^l(S, m)$ when S is drawn randomly from SIMATTEMPT^l.

Lemma 2 *The distributions* $\text{PATCH}^l(m)$ *and* $\text{ATTEMPT}^l(m)$ *are computationally indistinguishable.*

Proof: Let A be the event that at least one attempt has $s = 0$ and $c \neq d$. The probability that A occurs in PATCH^l is 0 and for A to occur in $\text{ATTEMPT}^l(m)$ it must be the case, that $\mathcal{D}(C) = M$ in one of the successful attempts, where C and M are chosen independently and M chosen uniformly. The probability is therefore certainly less than $\frac{a}{2|\mathcal{M}|}$. Let $\text{PATCH}^l_i(m)$ and $\text{ATTEMPT}^l_i(m)$ for $i = 0, 1, \ldots, a$ be the distributions of $\text{PATCH}^l(m)$ resp. $\text{ATTEMPT}^l(m)$ under the condition that exactly i successful attempts occurred. By the computational distance between two distributions, we mean the maximal advantage with which they can be distinguished in polynomial time. Let δ_i be the computational distance between $\text{ATTEMPT}^l_i(m)$ and $\text{PATCH}^l_i(m)$. In both distributions f and b are uniformly random and $f = m \oplus b$. The remaining values occurring in the distributions are those chosen from ATTEMPT and PATCH. Therefore, by lemma 1 and independence of the attempts we have that $\delta_0 = 0$ and δ_1 is negligible. By independence of the attempts we further more have that $\delta_{i+1} = \delta_1 + \delta_i$, so $\delta_i = i\delta_1$. Assuming that A does not occur, the probability p_i that exactly i successful attempts occur is the same in both distributions, namely $\binom{a}{i}(\frac{1}{2})^i(\frac{1}{2})^{a-i}$. The total computationally distance given that A does not occur is therefore bounded by $\sum_{i=1}^{a} p_i\delta_i \leq 2^{-a}\delta_1 \sum_{i=1}^{a} i\binom{a}{i} = 2^{-a}\delta_1 a 2^{a-1} = \frac{a}{2}\delta_1$. It then follows by applying the triangle inequality, that the total computational distance is bounded by $\frac{a}{2}(\delta_1 + |\mathcal{M}|^{-1})$, which is certainly negligible in k. ■

[6]We can safely ignore the negligible few case where less than l successful attempts occur.

4.3 The simulator

The overall strategy of the simulator will consist of three phases. First a pre-processing phase, where the simulator prepares a set of data to simulate the communication in an execution of the real life protocol by drawing an element S from SIMATTEMPT^l. This provides values for all communication, but especially leaves the values of c and d open in the successful attempts. The pre-processing phase is followed by the oblivious simulation phase, where the simulator simulates a real-life execution by handing out the appropriate pre-processed communication values to the adversary. The oblivious simulation phase continues until the first corruption request from the adversary occurs. After a party P_i (S or R) has been corrupted by the adversary the patched-simulation phase starts. On corruption the adversary should receive the random bits used by P_i. To do this we obtain the communicated message $m \in \boldsymbol{B}^l$ of the ideal evaluation, by corrupting P_i in the ideal evaluation and patch S using m ($\text{PATCH}^l(S, m)$). After patching the simulator hands out the appropriate values to the adversary. Corrupting any party P_i provides the simulator with the message m of the ideal evaluation and thus with *all* information of the ideal evaluation. As the patched values handed to the adversary is computationally indistinguishable from those of a real life execution by lemma 2 and because they are consistent with the value m returned by the ideal-evaluation, the simulator can now exactly (or at least computationally indistinguishably) simulate the remaining uncorrupted party by using the appropriate pre-processed values as a starting value of variables that are in play at the time of corruption and simply run the uncorrupted party according to the full parallel protocol. As the starting values are computationally indistinguishable from those of a real life execution and all parties including the adversary are polynomially time bounded, so will the result of the simulation be, and the adversary will eventually return a value computationally indistinguishable from the value it would have returned after engaging in a real-life execution. Considering an environment the same reasoning holds. This implies, that the protocol is 2-adaptively secure.

Theorem 1 *If simulatable public-key systems exists, then strong non-committing encryption schemes exists.*

5 Implementations

The following theorem provides the first example of a simulatable public-key system.

Theorem 2 *The ElGamal public-key system is simulatable under the Decisional Diffie-Hellman(DDH) assumption.*

Proof: See appendix C. ■

5.1 Trapdoor One-Way Permutations

Before presenting the next example of a simulatable public-key system, we define the concept of a simulatable collection of one-way trapdoor permutations and prove that the existence of such a collection implies the existence of simulatable public-key systems.

We first recall the standard definition of collections of trapdoor permutations:

Definition 4 (Collection of one-way trapdoor permutations) *The 4-tuple $(I, F, \mathcal{G}, \mathcal{X})$ is a collection of one-way trapdoor permutations with security parameter k, if I is an infinite index set, $F = \{f_i : D_i \to D_i\}_{i \in I}$ is a set of permutations, the index/trapdoor generator $\mathcal{G}$ and the domain generator $\mathcal{X}$ are probabilistic polynomial time algorithms, and the following hold:*

Easy to generate and compute *$\mathcal{G}$ generates pairs of indices and trapdoors, i.e. $(i, t_i) \leftarrow \mathcal{G}(1^k)$, where $i \in I \cap \boldsymbol{B}^{p(k)}$, for some fixed polynomial $p(k)$. Furthermore, there is a polynomial time algorithm which on input $i, x \in D_i$ computes $f_i(x)$.*

Easy to sample domain *$\mathcal{X}$ samples elements in the domains of the permutations, i.e. $x \leftarrow \mathcal{X}(i)$, where x is uniformly random in D_i.*

Hard to invert *For $(i, t_i) \leftarrow \mathcal{G}(1^k)$, $x \leftarrow \mathcal{X}(i)$ and for any probabilistic polynomial time algorithm A the probability that $A(i, f_i(x)) = x$ is negligible in k.*

But easy with trapdoor *There is a polynomial time algorithm which on input $i, t_i, f_i(x)$ computes x, for all $x \in D_i$.*

The next definition, of simulatable collections, is built along the lines of the definition of simulatable public-key systems. It basically defines a collection of one-way trapdoor permutations where in addition it is easy to generate a permutation f in the collection without getting to know the trapdoor. For technical reasons, we also need the property that if an element y was generated by choosing x randomly and setting $y = f(x)$, one can always claim that y was chosen by sampling the domain of f directly. This amounts to being able to compute from y and backwards through the sampling algorithm to find random bits leading to y being sampled. Therefore we call this invertible sampling.

Invertible sampling is trivial if the domain of f is, for instance, the set of k-bit strings and sampling is done in the natural way. But it may in general be an extra requirement which, however, seems to be necessary for any application of the kind we consider here. It is easy to construct artificial examples without the invertible sampling property, but all reasonable examples we know of have this property.

Definition 5 (Simulatable collection of one-way trapdoor permutations) *Let $(I, F, \mathcal{G}, \mathcal{X})$ be a collection of one-way trapdoor permutations with security parameter*

k. We say that $(I, F, \mathcal{G}, \mathcal{X})$ *is a* simulatable collection of one-way trapdoor permutations *with* oblivious index generator $\tilde{\mathcal{G}}$, index faking algorithm $\tilde{\mathcal{G}}^{-1}$, *and* domain faking algorithm $\mathcal{X}^{-1}$, *if* $\tilde{\mathcal{G}}$, $\tilde{\mathcal{G}}^{-1}$, *and* $\mathcal{X}^{-1}$ *are probabilistic polynomial time algorithms and the following holds:*

Oblivious index generation *Let* $r \leftarrow \mathcal{R}_{\mathcal{G}}$, $(i, t_i) \leftarrow \mathcal{G}(r)$, $r' \leftarrow \tilde{\mathcal{G}}^{-1}(i)$ *and* $\tilde{r} \leftarrow \mathcal{R}_{\tilde{\mathcal{G}}}$, $\tilde{i} \leftarrow \tilde{\mathcal{G}}(\tilde{r})$. *Then the distributions* (r', i) *and* $(\tilde{r}, \tilde{i})$ *are computationally indistinguishable.*

Invertible sampling *Let* $r_0 \leftarrow \mathcal{R}_{\mathcal{X}}$, $x \leftarrow \mathcal{X}(i, r_0)$, *and* $r_1 \leftarrow \mathcal{X}^{-1}(i, x)$. *Then the distributions* (r_0, x) *and* (r_1, x) *are computationally indistinguishable.*

Given F a simulatable collection of one-way trapdoor permutations one can construct a semantically secure public-key system using the construction in [11] for building a semanticly secure encryption scheme from a collection of one-way trapdoor permutations. We review the construction here. Let B be a hard-core predicate of the collection of one-way trapdoor permutations. If no such B is known one can construct a new simulatable collection of one-way trapdoor permutations following the construction in [11]. The key generator is set to be $\mathcal{K} = \mathcal{G}$, i.e. for $(i, t_i) \leftarrow \mathcal{G}$ we set $(P, S) = (i, t_i)$. The message space can be set to $\mathcal{M} = \boldsymbol{B}^{p(k)}$ for any polynomial $p(k)$ and the cipher-text space for $P = i$ is $\mathcal{M} \times D_i$, where D_i is the domain of f_i. Encryption and decryption is given by Fig. 3. Finally set $\tilde{\mathcal{K}} = \tilde{\mathcal{G}}$ and $\tilde{\mathcal{K}}^{-1} = \tilde{\mathcal{G}}^{-1}$.

```
proc E_P(m ∈ M) ≡                proc D_S(c, c_|c|) ≡
  x ← X(P);                        for i = |c| − 1 to 0 do
  c_0 ← x;                           c_i ← f_S^{-1}(c_{i+1});
  for i = 1 to |m| do                b_{i+1} ← B(c_i);
      c_i ← f_P(c_{i−1});          od;
      b_i ← B(c_{i−1});            p ← b_1 b_2 ... b_|c|;
  od;                              m ← c ⊕ p;
  p ← b_1 b_2 ... b_|m|;           exit m
  c ← m ⊕ p;
  exit (c, c_|m|)
```

Figure 3: Encryption and decryption

Theorem 3 *Let* $\mathcal{F} = (I, F, \mathcal{G}, \mathcal{X})$ *be a simulatable collection of one-way trapdoor permutations and let* $E_{\mathcal{F}} = (\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{M})$ *be the public-key system described above. Then* $E_{\mathcal{F}}$ *is simulatable.*

Proof: Oblivious key generation follows directly. To pick a cipher-text obliviously for a given key P generate $x \leftarrow \mathcal{X}(P)$ and $m \leftarrow \mathcal{M}$ and let $C = (m, x)$. This will be distributed exactly as $C \leftarrow \mathcal{E}_P(m')$ for $m' \leftarrow \mathcal{M}$. To fake a real encryption to look oblivious we can use $\mathcal{X}^{-1}$. In [11] is is proven that the scheme is semantically secure. ■

We proceed to construct a simulatable collection of one-way trapdoor permutations based on RSA. The security will be based on the following assumption 1.

Assumption 1 *Let* $I = \{(n,e)|n = pqr,\ p, q$ *are primes and* $|p|, |q| \geq k,\ |n| = k \log k,$ *and* $n < e < 2n$ *is a prime*$\}$. *Here,* k *is as usual the security parameter. For* $(n,e) \in I$ *let* $t_{(n,e)} = d$ *where* $ed = 1 \mod \phi(n)$. *Let* $f_{(n,e)} : \boldsymbol{Z}_n^* \to \boldsymbol{Z}_n^*, x \mapsto x^e \mod n$. *Then* $F = \{f_i\}_{i \in I}$ *is a collection of one-way trapdoor permutations.*

Note that $e' = (e \mod \phi(n)) \in \boldsymbol{Z}_{\phi(n)}^*$ as e is prime and that F is therefore certainly a collection of permutations. Secondly the Chinese Remainder theorem implies that inverting $f_{(n,e)}$ is no easier than inverting the RSA functions over $\boldsymbol{Z}_{pq}^*$, where both factors have $\Omega(k)$ bits. Further more for fixed n and random e the value $e \mod \phi(pq)$ is random over a large subset of $\boldsymbol{Z}_{\phi(pq)}^*$. The above assumption therefore reduces to more traditional forms of the RSA assumption.

We now proceed to describe the index and trapdoor generation algorithms $\mathcal{G}$ and $\tilde{\mathcal{G}}$ of a simulatable collection by the programs in Fig. 4. The idea is that we set $l = \log k$, and generate l moduli $n_1, ..., n_l$ and a public exponent e. The domain for this index $i = (e, n_1, \ldots, n_l)$ will be $D_i = \prod_{j=1}^{l} \boldsymbol{Z}_{n_j}^*$ and the corresponding permutation will be $f_{(e,n_1,\ldots,n_l)}(x_1, \ldots, x_l) = (x_1^e \mod n_1, \ldots, x_l^e \mod n_l)$.

proc $\mathcal{G}(1^k)$ $\equiv$
  $e \leftarrow$ primes$[2^{k \log k}, 2^{k \log k+1}]$;
  for $j = 1$ **to** l **do**
    $n_j \leftarrow [2^{k \log k-1};\ 2^{k \log k}]$
      with known factorisation;
    $d_j \leftarrow e^{-1} \mod \phi(n_j)$ **od**;
  $i \leftarrow (e, n_1, \ldots, n_l)$;
  $t_i \leftarrow (d_1, \ldots, d_l)$;
  exit (i, t_i)

proc $\tilde{\mathcal{G}}(1^k)$ $\equiv$
  $e \leftarrow$ primes$[2^{k \log k}, 2^{k \log k+1}]$;
  for $j = 1$ **to** l **do**
    $n_j \leftarrow [2^{k \log k-1};\ 2^{k \log k}]$
      with unknown factorisation **od**;
  $i \leftarrow (e, n_1, \ldots, n_l)$;
  exit i

Figure 4: Index and trapdoor generation

A note on the algorithms is necessary. By "$n_j \leftarrow [2^{k \log k-1};\ 2^{k \log k}]$; known factorisation" we mean, that n_j should be picked uniformly random from $[2^{k \log k-1};\ 2^{k \log k}]$ and that we should obtain its prime factorisation at the same time. In [1] it is shown how to do this in probabilistic polynomial time - the algorithm obtains a distribution negligible close to the uniform distribution which suffice for our application. By "$e \leftarrow$ primes$[2^{k \log k}, 2^{k \log k+1}]$" we mean that e should be picked uniformly random as a prime between $2^{k \log k}$ and $2^{k \log k+1}$. To fake an index $i = (e, n_1, \ldots, n_l)$ generated by $\mathcal{G}$ let $\tilde{\mathcal{G}}^{-1}(i)$ return $(n_1, \ldots, n_l)$ and random bits computationally indistinguishable from those used to construct e. This will be distributed computationally indistinguishable from the random bits used by $\tilde{\mathcal{G}}$ to generate the same index. How to produce random bits computationally indistinguishable from those used to pick e of course depends on how e is picked. Say we pick e by drawing random numbers in $[2^{k-1};\ 2^k]$ until we get a number that test to primality by some probabilistic test.

We will then have to reconstruct, from e, a distribution similar to the prefix of numbers that were not primes. This can trivially be done be drawing random numbers until a prime is found and use the prefix of non-primes. That the oblivious index generation property is fulfilled is then obvious.

Since e is relatively prime to all n_j our functions are indeed permutations and are invertible in probabilistic polynomial time using the trapdoor information $t_i = (d_1, \ldots, d_l)$. We pick a uniformly random element x from D_i by picking a uniformly random element x_j from each $\boldsymbol{Z}^*_{n_j}$. These elements should be chosen in a way that allows us to construct a domain faking algorithm $\mathcal{X}^{-1}$. One way is to pick uniformly random elements from $\boldsymbol{Z}_{n_j}$ until an element from $\boldsymbol{Z}^*_{n_j}$ is found. We can then fake given just the last element - cf. the above discussion of $\tilde{\mathcal{G}}^{-1}$. This gives us $\mathcal{X}^{-1}$, which obviously fulfils the invertible sampling property.

Theorem 4 *Under assumption 1, the set $SRSA = \{f_i : D_i \rightarrow D_i\}$ is a simulatable collection of one-way trapdoor functions.*

Proof: As described the oblivious index generation property and the invertible sampling property are evident. Proving one-wayness remains. In [10] the probability that the i'th largest primefactor of a random number n is larger than n^c for a given constant c is investigated. It is shown to approach a constant as n approaches infinity. In particular, the probability that the second largest primefactor is smaller than n^c is approximately linear for small c, in fact it is about $2c$ for $c \leq 0.4$. It follows that the probability that a number of length $k \log k$ bits has its second largest prime factor shorter than k bits is $O(1/\log k)$. Hence the probability that there does not exist $j \in \{1, \ldots, l\}$ such that $(n_j, e) \in I$, where I is the index set of assumption 1, is $O((\frac{1}{\log k})^{\log k})$ and so is negligible. This implies that $SRSA$ is a collection of one-way trapdoor functions. ■

5.2 Doing without Oblivious Index Generation

5.2.1 A problem with a subprotocol from [4] and its solution

In [4] a non-committing encryption scheme was built consisting of two stages. The first phase is a key generation protocol which is intended to create a situation, were players S and R share two trapdoor permutations from what is called a common-domain trapdoor system. Moreover, S knows exactly one of the corresponding trapdoors, and if S remaines honest in this phase, a simulator is able to make a simulated computation, where both trapdoors are learned and which can later (in case S is corrupted) be convincingly patched to look as if either of the trapdoors were known to S. One immediate consequence is that the adversary must not know which of the two trapdoors is known to S, before corrupting S.

The key generation requires participation of all n parties of the protocol and proceeds as follows: Each player P_i chooses at random two permutations (g_0^i, g_1^i) and send these to S. Next S chooses $c = 0$ or 1 at random, and we execute the oblivious

transfer (OT) protocol of [9] with P_i as sender using the trapdoors of (g_0^i, g_1^i) as input and S as receiver using c as input, and such that S receives the trapdoor of g_c^i. The OT protocol of [9] has a non-binding property that allows a simulator to learn both trapdoors when it is playing S's part and later to claim that either trapdoor was received.

In the above, there is no guarantee that P_i really uses the trapdoors of (g_0^i, g_1^i), as input to the OT, but, as pointed out in [4] one may assume that the trapdoor of a permutation consists of all inputs required to generate it so that S can verify what he receives. Finally, S publishes the subset A of players from whom he got correct trapdoors, and we define f_0 to be the composition of the permutations $\{g_0^i\}_{i\in A}$ in some canonical order, and similarly for f_1. The permutations can be composed because it is assumed in [4] that they can be constructed to all have the same domain.

There is, however, a problem in this protocol: if S is still honest, but P_i is corrupt, the adversary may choose to let P_i use as inputs to the OT a correct trapdoor for f_a^i but garbage for f_b^i. Then, when the adversary sees the set A, he can conclude that $c = a$ if $i \in A$, and $c = b$ otherwise. As we found above, this is a piece of information that the adversary should not be able to get.

Fortunately, this problem is simple to solve: we require players to also provide a zero-knowledge proof that they input correct information to the OT. Since this is an NP statement, this is always possible [9]. This will imply that except with negligible probability, P_i will have to supply correct trapdoors to both permutations or be disqualified. Normally, the use of such ZK proofs lead to problems against adaptive adversaries because of the rewinding needed to simulate the proofs. However, in this protocol, it happens to be the case that the simulator never needs to "prove" a statement for which it doesn't know a witness, and so rewinding is not needed. More details on this can be found in Appendix A.

5.2.2 Using the Key Generation in Our Protocol

After fixing the bug above, we do a slight modification of the key generation such that it can work for *any* collection of one-way trapdoor permutations with invertible sampling and we can then use this key-generation phase instead of the first round of our protocol. We therefore do not need the oblivious index generation property from definition 5. The modification is as follows:

Having executed all the OT's, S defines a permutation f_c for which he knows the trapdoor by setting $f_c(x_1, \ldots, x_l) = (g_c^1(x_1), \ldots, g_c^l(x_l))$, where $g_c^1, \ldots, g_j^l$ are the permutations received for which he learned the trapdoor in the OT. A permutation f_{1-c} for which he does not know the trapdoor is constructed similarly from those permutations received for which he did not learn the trapdoor. He then sends f_0, f_1 to R. When doing encryption in the standard way based on f_0, we will let the hard-core bit function for f_0 be the xor of the hard-core function for the collection from which the $g_0^1, \ldots g_0^l$ permutations were chosen. Similarly for f_1.

Note that an adversary will not know the entire trapdoors of f_0, f_1 (unless all parties are corrupted), so using the non-committing property of the OT scheme a simulator can learn all trapdoors and claim that either f_0 or f_1 is the permutation for which all trapdoor information is known to S.

This is exactly the scenario that the first round of our protocol described earlier creates. Thus, from here our communication phase can then be used to complete the non-committing encryption protocol.

The sketched key-generation phase is secure as long as at least one party is not corrupted, but in contrary to the approach were simulatable public-key systems are used all trapdoor information is explicitly present somewhere in the network, so the protocol is not resilient against an adversary that corrupts all parties.

Theorem 5 *If there exists collections of one-way trapdoor functions with invertible sampling, then non-committing encryption schemes exists.*

We note that the OT protocol which we use as an essential tool is itself based on one-way trapdoor permutations. Moreover, in order for the OT to be non-committing, the permutations must have invertible sampling. This property is also necessary in the original key generation protocol from [4], where also a common domain property was needed, so assuming only invertible sampling is a weaker assumption.

References

[1] ERIC BACH. How to generate factored random numbers. In S. Rao Kosaraju, editor, *SIAM Journal on Computing*, volume 17, pages 179–193. Society for Industrial and Applied Mathematics. Philadelphia. Pennsylvania, 1988.

[2] Donald Beaver. Foundations of secure interactive computation. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 377–391. Springer-Verlag, 1991.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Proc. ACM STOC '88*, pages 1–10.

[4] R. Canetti, U. Feige, O. Goldreich, and M. Naor. Adaptively secure computation. In *28th Symposium on Theory of Computing (STOC)*, pages 639–648. ACM, 1996.

[5] Ran Canetti. Security and composition of multi-party cryptographic protocol. Obtainable from the Theory of Cryptography Library, august 1999.

[6] D. Chaum, C. Crèpeau, and I. Damgård. Multi-party unconditionally secure protocols. In *Proc. of ACM STOC '88*.

[7] A. De-Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *33rd FOCS*, pages 427–436, 1992.

[8] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. of ACM STOC '87*, pages 218–229.

[9] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but the validity of the assertation, and a methodology of cryptographic protocol design. In *Proc. IEEE FOCS'86*, pages 174–187.

[10] Donald E. Knuth and Luis Trabb Pardo. Analysis of a simple factorization algorithm. *Theoretical Computer Science 3*, pages 321–348, 1976.

[11] M.Blum and S.Goldwasser. An efficient probabilistic public-key encryption scheme which hides all partial information. In G.R.Blakley and D.C.Chaum, editors, *CRYPTO 84*, volume 196 of *Lecture Notes in Computer Science*, pages 289–302. Springer, 1985.

[12] Silvio Micali and Phillip Rogaway. Secure computation. In J. Feigenbaum, editor, *Advances in Cryptology - CRYPTO '91*, volume 576 of *Lecture Notes in Computer Science*, pages 392–404. Springer-Verlag, 1991.

[13] A. Yao. Protocols for secure computation. In *Proc. IEEE FOCS '82*, pages 160–164.

A A Simulator for Non-committing Encryption with Corrected Key Generation

We assume in this section that the key generation protocol from [4] is corrected to include zero-knowledge proofs that correct input is supplied to the OT protocols as described above. We then use the key generation in stead of the first round in our attempt protocol. Let $g_c^1, \ldots, g_c^l$ be the permutations that was correctly received and for which the corresponding trapdoor was received. From these permutations S defines a new permutation f_c, where $f_c(x^1, \ldots, x^l) = (g^1(x^1), \ldots, g^l(x^l))$. Let B be a hard-core predicate for the collection of trapdoor permutations used. Then $B(x^1, \ldots, x^l) = \bigoplus_{i=1}^{l} B(x^i)$ is a hard-core predicate for f_c. Let $x^1, \ldots, x^l$ be uniformly random and let $X(g^i, x^i, n) = B(x^i)B(g^i(x^i))B((g^i)^2(x^i)) \ldots B((g^i)^{n-1}(x^i))$ be the usual pseudo-random string generated from x^i. Then the encryption of $m \in \boldsymbol{B}^*$ under f_c using the above hard-core predicate is seen to be $((g^1)^{|m|}(x^1), \ldots, (g^l)^{|m|}(x^l)), m \oplus X(g^1, x^1, |m|) \oplus \ldots \oplus X(g^l, x^l, |m|)$. Similar for f_{1-c}. After defining these two permutations we use the original protocol to complete the non-committing encryption.

For sake of simplicity we will analyse the sequential protocol, where independent attempts are carried out until a successful attempt occurs. The arguments extends

directly to the parallel protocol. We sketch how simulation of the sequential protocol can be done against an adaptive adversary. As before all failed attempts are simulated completely honestly by following the real-life protocol. To recapture, this is possible as the result of these attempts are never used later and is therefore independent of the input of all parties. In the simulation before each attempt, we draw a uniformly random bit s to determine whether the attempt should be successful or not. Obviously, if any of S or R is corrupted during a failed attempt, we can obtain the message bit $m \in \boldsymbol{B}$ of the ideal evaluation and then follow the real-life protocol from that point. The distribution of the simulation will then be *exactly* equal to the real-life execution.

If we reach the successful attempt without S or R being corrupted, we will have to simulate that attempt too. Recall that the player acting as the receiver in the OT's is called S, whereas R is the player that wants to eventually receive a message from S in a non-committing fashion. Each player P_i acts as sender in an OT protocol.

Now, as long as R, S are still honest, we do as follows

- If P_i is honest at the start of the OT, we choose permutations "on behalf of P_i" with known trapdoors and simulate in the obvious way. If some P_i becomes corrupt, we just give to the adversary the trapdoors we chose (which will be fine because they do not depend on P_i's input)

- If P_i is corrupt, we play S's part, and choose to learn both trapdoors. Since P_i must prove he's providing good data, this is guaranteed to either gives us both trapdoors, or disqualify P_i (except with negligible probability).

If R or S is corrupted, we learn as above the message to be sent by corrupting in the ideal process, then we patch the simulation created so far to be consistent with this, using our knowledge of all trapdoors and the non-binding property of the OT protocol - details can be extracted from the description of patching in the simulation of the original attempt protocol.

We now simulate the last part of the non-committing encryption in the same way as described earlier, exploiting that we know all trapdoors.

Note that in simulating the key-generation phase of the successful attempt the only deviation from the real-life protocol is that both trapdoors are learned by S in the OTs. If S is later corrupted, the patching of his view to make it look as either trapdoor was learned only requires an application of the invertible sampling algorithm, so using that all failed attempts are simulated equal to the real-life execution it is immediate, that until, but not including, the round, where R sends the (M, C) pairs in the successful attempt, the distribution of the simulation and the distribution of the real-life protocol is indistinguishable even after a corruption of either party. I.e. if the simulator learns m before that round it can certainly carry through the simulation successfully.

This is an important point to note, so lets restate it differently. An adversary can only distinguish a simulation from a real-life execution if it does not corrupt S and R until after the round, where R sends the (M, C) pairs to S.

There might however be a possibility for an adversary to distinguish if R or S is corrupted in the successful attempt after the (M, C) pairs are send (or post-corrupted). In the simulation the adversary will see a pair (M, C) as the $(1-c)$ pair, where M is random and C is an encryption of M under the permutation f_{1-c} (we write $C = \mathcal{E}_{f_{1-c}}(M)$, suppressing the random coins)[7], but in a real execution (M_{1-c}, C_{1-c}) are independently chosen cipher- and plaintexts. Intuitively, since in this the adversary does not know the complete trapdoor of f_{1-c}, he cannot tell the difference.

To prove this, suppose that there exists a real-life adversary $\mathcal{A}$, an environment $\mathcal{Z}$, and an input value m, such that simulation and real execution is distinguishable. We prove that this contradicts the one-wayness of the one-way trapdoor permutations used in the implementation. Specifically we use the real-life adversary $\mathcal{A}$, the environment $\mathcal{Z}$, and the input value m to build a distinguisher $\mathcal{D}$, that will distinguish the random variables $(g, z, y \leftarrow \mathcal{E}_g(z))$ and $(g, z, y \leftarrow \mathcal{C}_g)$, where g is a random one-way trapdoor permutation from the collection used in the protocol, z is a random message, $\mathcal{E}_g(z)$ denotes the usual encryption of x under g, and $y \leftarrow \mathcal{C}_g$ denotes an obliviously chosen cipher-text (using the invertible sampling property of g.)

As noted above we can without lose of generality assume that the adversary $\mathcal{A}$ does not corrupt S or R until after the (M, C) pairs have been send. We construct $\mathcal{D}$ as follows. Given input g, z, y we run the simulator algorithm, except that we place the permutation g at random as one of those permutation, for which S will not learn the corresponding trapdoor - we know the input m to the protocol, so we are able to do this. I.e., we pretend it was chosen by one of the players P_i as g^i_{1-c}. We hope, that that party is not corrupted at all. This happens with probability at least $\frac{1}{n}$ as g is distributed exactly as the permutations in the real-life execution and the adversary can corrupt at most $n-1$ parties. Now observe that the proof that the trapdoor of g is input to the OT is executed at a point where S is still honest (i.e. before the round where R sends the two (M, C) pairs.) Therefore we can simulate it by honest verifier simulation (which does not require rewinding, nor knowledge of the trapdoor of g).

If the party, that was given g was not corrupted, then there is some j, such that $g = g^j_{1-c}$. Now construct a pair (M_{1-c}, C_{1-c}) by setting $M_{1-c} = z$ and letting C_{1-c} be the result of encrypting z as described above with the exception, that we use y as the contribution from $g = g^j_{1-c}$ to the encryption.[8] From the description of the

[7]The following analysis can be extended to consider the random bits using the invertible sampling property.

[8]We have that $y = g^{|z|}(x), z \oplus X(g, x, |z|)$, where x is some random value from the domain of g. We can therefore construct a full encryption by xoring the $X(g^i, x^i, |z|)$ value of the remaining permutations into $z \oplus X(g, x, |z|$ and including the remaining $(g^i)^{|z|}(x^i)$ values in the encryption of z under g.

encryption algorithm, it follows that $C_{1-c} = \mathcal{E}_{f_{1-c}}(M_{1-c})$ precisely if $y = \mathcal{E}_g(z)$ and otherwise (M_{1-c}, C_{1-c}) is distributed exactly as $(M, C \leftarrow \mathcal{C}_{f_{1-c}})$ because a uniformly random string is xored into the encryption. We now place (M_{1-c}, C_{1-c}) in the simulation as the $(1-c)$ pair send by R. Again this is possible because we know m.

By the assumptions, S and R will remain uncorrupted until we have send (M_{1-c}, C_{1-c}) and the party holding g will remain totally uncorrupted with non-negligible probability. If both assumptions hold, we show the view we generated to the adversary and the environment and output whatever it outputs. Now, if the pair is $(M_{1-c}, C_{1-c}) = (M_{1-c}, \mathcal{E}_{f_{1-c}}(M_{1-c}))$, then this view is indistinguishable from that of the simulator (it is only indistinguishable from rather than equal to, because of the honest verifier simulation of the zero-knowledge proof in the OT). On the other hand, if (M_{1-c}, C_{1-c}) are independent we get something indistinguishable from the real view (now only indistinguishable because of the use of invertible sampling for the OT and the patching of views and for the reason mentioned above). The adversary/environment will therefore distinguish the two views with non-negligible probability. Thus we have built an efficient algorithm that distinguishes the two kinds of pairs, contradicting the assumption that f comes from a one-way trapdoor collection.

B The full Parallel Protocol

The full parallel protocol is described in Fig. 5. The one-time pad is computed from (c, s) and (d, s) in the following manner. Except for negligible many cases $s = c \oplus d$, so R can compute $c = d \otimes s$. The one-time pad b is then set to be the bitstring consisting of the $l(k)$ first bits bits c_i of c, where $s_i = 0$.

C Results

Proof: (of theorem 2) Recall that a public key is a triple (p, g, h), where p is a prime such that the discrete log problem in $\boldsymbol{Z}_p$ is intractable, $\langle g \rangle = \boldsymbol{Z}_p^*$, and $h = g^x$ for some random x which is the private key. Now simply let the oblivious key generator pick h directly in $\boldsymbol{Z}_p^*$ without learning its discrete log base g and let $\tilde{\mathcal{K}}^{-1}(p, g, h, x) = (p, g, h)$. Then the oblivious public-key generation property is trivially fulfilled. A message $x \in \boldsymbol{Z}_p^*$ is encrypted as (g^k, xh^k), where k is chosen uniformly random in $\boldsymbol{Z}_{p-1}$. It is obvious, that a cipher-text can be generated obliviously as (y_1, y_2), where y_1 and y_2 are picked uniformly random and independent from $\boldsymbol{Z}_p^*$. A "real" message is thereby trivially fakable. ■

In [2] Beaver proposed an encryption protocol based on the property used in the above proof that one can indistinguishably pick an element with or without learning its corresponding discrete log. The protocol in [2] is based on the Diffie-Hellman key-exchange protocol. Two exchanges are carried out in parallel and each party

S		**R**
$m \in \boldsymbol{B}^{l(k)}$		Λ
$a \leftarrow 4l(k)$		$a \leftarrow 4l(k)$
$c \leftarrow \boldsymbol{B}^a$		$d \leftarrow \boldsymbol{B}^a$
for $j = 1$ **to** a **do**		**for** $j = 1$ **to** a **do**
$r^j_{c_j} \leftarrow \mathcal{R}_{\mathcal{K}}$		$e^j_{d_j} \leftarrow \mathcal{R}_{\mathcal{E}}$
$r^j_{1-c_j} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}$		$e^j_{1-d_j} \leftarrow \mathcal{R}_{\mathcal{C}}$
$(P^j_{c_j}, S^j_{c_j}) \leftarrow \mathcal{K}(r_c)$		**od**
$P^j_{1-c_j} \leftarrow \tilde{\mathcal{K}}(r_{1-c})$		
od		
	$\xrightarrow{P}$	
		for $j = 1$ **to** a **do**
		$M^j_{d_j} \leftarrow \mathcal{M}_{P^j_{d_j}}$
		$M^j_{1-d_j} \leftarrow \mathcal{M}_{P^j_{1-d_j}}$
		$C^j_{d_j} \leftarrow \mathcal{E}_{P^j_{d_j}}(M^j_{d_j}, e^j_{d_j})$
		$C^j_{1-d_j} \leftarrow \mathcal{C}_{P^j_{1-d_j}}(e^j_{1-d_j})$
		od
	$\xleftarrow[C]{M}$	
for $j = 1$ **to** a **do**		
$s^j \leftarrow \begin{cases} 0 & \text{if } \mathcal{D}_{S^j_{c_j}}(C^j_{c_j}) = M^j_{c_j} \\ 1 & \text{otherwise} \end{cases}$		
od		
compute $b \in \boldsymbol{B}^{l(k)}$ from (c, s)		
$f \leftarrow b \otimes m$		
	$\xrightarrow{s,f}$	
		compute $b \in \boldsymbol{B}^{l(k)}$ from (d, s)
Λ		$f \otimes b$

Figure 5: The full parallel protocol.

randomly chooses one of the exchanges to gable. This garbling is detectable by the parties and in the lines of our proposal they can securely use this bit as a one-time pad when they make the same choice of exchange to gable. The protocol is proven to be adaptively secure in the computational setting relative to DDH, though the model used in [2] is different from the one in [5] in that it e.g. does not consider an environment. The proof can however easily be extended to fit the definition of security in [5] following the lines of the proof in Chap. 4.2.

Lemma 3 *Let $(\mathcal{K}, \mathcal{E}, \mathcal{D}, \mathcal{M})$ be a simulatable public-key system. Then the distribution*

$$(\tilde{r} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}, \tilde{P} \leftarrow \tilde{\mathcal{K}}(\tilde{r}), M \leftarrow \mathcal{M}_{\tilde{P}}, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_{\tilde{P}}(e)) \tag{3}$$

and the distribution

$$(r', P, M \leftarrow \mathcal{M}_P, e', C \leftarrow \mathcal{E}_P(M, e)) \tag{4}$$

where $r \leftarrow \mathcal{R}_{\mathcal{K}}$, $(P, S) \leftarrow \mathcal{K}(r)$, $r' \leftarrow \tilde{\mathcal{K}}^{-1}(P)$ *and* $e \leftarrow \mathcal{R}_{\mathcal{E}}$, $e' \leftarrow \mathcal{C}^{-1}(C, P)$ *are computationally indistinguishable.*

Proof: By the oblivious public-key generation property the distributions $(\tilde{r} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}, \tilde{P} \leftarrow \tilde{\mathcal{K}}(\tilde{r}))$ and (r', P), where $r \leftarrow \mathcal{K}_{\mathcal{K}}$, $(P, S) \leftarrow \mathcal{K}(r)$, and $r' \leftarrow \tilde{\mathcal{K}}^{-1}(P)$ are computationally indistinguishable. Since the function $\lambda(r, P) = (r, P, M \leftarrow \mathcal{M}_P, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_P(e))$ can be computed in probabilistic polynomial time, by applying that to the computationally indistinguishable distributions, we get that

$$(\tilde{r} \leftarrow \mathcal{R}_{\tilde{\mathcal{K}}}, \tilde{P} \leftarrow \tilde{\mathcal{K}}(\tilde{r}), M \leftarrow \mathcal{M}_{\tilde{P}}, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_{\tilde{P}}(e)) \tag{5}$$

and

$$(r', P, M \leftarrow \mathcal{M}_P, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_P(e)) \tag{6}$$

are computationally indistinguishable. Now using the oblivious cipher-text generation property we have that the distributions $(P, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_P(e))$ and $(P, e', C \leftarrow \mathcal{E}_P(M_1, e)$, where $e \leftarrow \mathcal{R}_{\mathcal{E}}$, $e' \leftarrow \mathcal{C}^{-1}(C, P)$, and $M_1 \leftarrow M$ are computationally indistinguishable. Applying $\lambda(P, e, C) = (r' \leftarrow \tilde{\mathcal{K}}^{-1}(P), P, M \leftarrow \mathcal{M}_P, e, C)$ we then get that the distributions

$$(r', P, M \leftarrow \mathcal{M}_P, e \leftarrow \mathcal{R}_{\mathcal{C}}, C \leftarrow \mathcal{C}_P(e)) \tag{7}$$

and

$$(r', P, M \leftarrow \mathcal{M}_P, e', C \leftarrow \mathcal{E}_P(M_1, e)) \tag{8}$$

are computationally indistinguishable.

Using similar argument and semantic security (8) is computationally indistinguishable from (4) which completes the proof. ■

Recent BRICS Report Series Publications

RS-00-6 **Ivan B. Damgård and Jesper Buus Nielsen.** ***Improved Non-Committing Encryption Schemes based on a General Complexity Assumption*****. March 2000. 24 pp.**

RS-00-5 **Ivan B. Damgård and Mads J. Jurik.** ***Efficient Protocols based on Probabilistic Encryption using Composite Degree Residue Classes*****. March 2000. 19 pp.**

RS-00-4 **Rasmus Pagh.** ***A New Trade-off for Deterministic Dictionaries*****. February 2000.**

RS-00-3 **Fredrik Larsson, Paul Pettersson, and Wang Yi.** ***On Memory-Block Traversal Problems in Model Checking Timed Systems*****. January 2000. 15 pp. To appear in** ***Tools and Algorithms for The Construction and Analysis of Systems: 6th International Conference*****, TACAS '00 Proceedings, LNCS, 2000.**

RS-00-2 **Igor Walukiewicz.** ***Local Logics for Traces*****. January 2000. 30 pp.**

RS-00-1 **Rune B. Lyngsø and Christian N. S. Pedersen.** ***Pseudoknots in RNA Secondary Structures*****. January 2000. 15 pp. To appear in** ***Fourth Annual International Conference on Computational Molecular Biology*****, RECOMB '00 Proceedings, 2000.**

RS-99-57 **Peter D. Mosses.** ***A Modular SOS for ML Concurrency Primitives*****. December 1999. 22 pp.**

RS-99-56 **Peter D. Mosses.** ***A Modular SOS for Action Notation*****. December 1999. 39 pp. Full version of paper appearing in Mosses and Watt, editors,** ***Second International Workshop on Action Semantics*****, AS '99 Proceedings, BRICS Notes Series NS-99-3, 1999, pages 131–142.**

RS-99-55 **Peter D. Mosses.** ***Logical Specification of Operational Semantics*****. December 1999. 18 pp. Invited paper. Appears in Flum, Rodríguez-Artalejo and Mario, editors,** ***European Association for Computer Science Logic: 13th International Workshop*****, CSL '99 Proceedings, LNCS 1683, 1999, pages 32–49.**