PLAINTEXT AWARENESS VIA KEY REGISTRATION

JONATHAN HERZOG, MOSES LISKOV, SILVIO MICALI
MIT LABORATORY FOR COMPUTER SCIENCE

Abstract. In this paper, we reconsider the notion of plaintext awareness. We present a new model for plaintext-aware encryption that is both natural and useful. We achieve plaintext-aware encryption without random oracles by using a third party. However, we do not need to *trust* the third party, as we still guarantee security against adaptive chosen ciphertext attacks in the case where the third party is dishonest. We show a construction that achieves this definition under general assumptions. We further motivate this achievement by showing an important and natural application: giving additional real-world meaningfulness to the Dolev-Yao model.

1. Introduction

In this paper, we put forward and implement a new notion of plaintext-aware encryption that is both natural and useful.

A Beautiful but Controversial Notion. As insightfully introduced by Bellare and Rogaway [4] (see also [2] for refinements), an encryption scheme is *plaintext-aware* (PA) if, whenever an adversary creates a ciphertext, he must "know" its corresponding plaintext.

Despite its natural appeal, PA encryption has been somewhat controversial for two main reasons:

1. *Plaintext awareness fundamentally relies on random oracles.*

   Not only do all known implementations of PA encryption use random oracles, but the very definition of plaintext awareness has, so far, crucially depended upon them. Random oracles are fundamentally abstract constructs. Though sometimes they can be realized algorithmically, no such hope exists here: traditional PA encryption requires the random oracle not only to be random, but also to be an *oracle.*

   A random oracle is in essence a trusted third party that interacts with the rest of us only in a very rigid way: if one puts a string x on a special query tape, it will write a random bit b_x on a special answer tape. This codified interface guarantees that even an adversary who purposely tries not to "know" what he is doing must be aware of his queries to the random oracle: after all he has to explicitly write each and every bit of x on the query tape! It is this elementary awareness that is cleverly exploited by Bellare and Rogaway to imply a much more sophisticated awareness: by looking at just the queries that an adversary makes to the random oracle during the computation of a ciphertext, one can easily deduce the underlying plaintext. The random oracle thus provides a magical "window" into the state of the encrypting algorithm, forcing it to disclose parts of its internal state.

   One can hardly fault the PA inventors for depending on random oracles: without any additional help plaintext awareness looks to be essentially impossible.
2. *Plaintext awareness has found no important and novel applications.*

   Plaintext-awareness is so strong a property that it immediately implies security against chosen-ciphertext attacks (CCA-2 security to be exact, in the notation of [2]): in essence, if the adversary already "knows" the answer that it will receive from a decryption oracle, then the oracle gives him no additional power.

   However, though very difficult to find and to prove, CCA-2 secure schemes were already known: they were constructed by Cramer and Shoup [7], under the decisional Diffie-Hellman assumption (yielding a very efficient scheme), and by Sahai [18] (improving on previous work of Naor and Yung [15]), under very general complexity assumptions. Thus, genuinely new applications of PA encryption, despite its intuitive great power, have been somewhat scarce.

Key words and phrases. Plaintext-aware encryption, Dolev-Yao, chosen-ciphertext security, public-key encryption.

Our Contributions. The main contributions of this paper are:

1. A new *definition* of PA encryption that does not use the random oracle.
2. An *implementation* of the new definition that is based on very general complexity assumptions.
3. A new and natural *application* of PA encryption that requires its full power.

To be sure, we still need to access a trusted third party, but our party is much more natural (in some sense, it is already there) and we access it only once rather than at every encryption.

The Essence of Our Definition. Our model is very simple: encryption is available only between users who have properly registered their public keys with a *registration authority,* and plaintext-awareness is guaranteed if this authority is honest.

This third-party model has several attractive features:

- *Safety:* The *plaintext awareness* of our scheme depends on the honesty of the registration authority, but not its *security*. Indeed, even if the registration authority collaborated with the adversary, our scheme is guaranteed to be CCA-2 secure.
- *Naturalness:* Trusting a registration authority is essentially already implicit in any actual implementation of public-key encryption. A public-key cryptosystem in fact requires a correct association between users and public keys. (Else, by publishing his own key under Alice's name, an adversary may easily read a message intended for Alice alone.) This correct association is guaranteed by the honesty of the key registration authority.

  Actually, in today's high-security applications, the registration authority also checks that a registrant really owns his public key (for instance, by asking the registrant to decrypt a random challenge). Else, after Bill Gates registers his own key, anyone could register that same key under his own name. Our key registration process is actually a natural enhancement of this very step.
- *Efficiency:* As we said, a random oracle is an abstraction, but it could be implemented by a trusted third party. However, in the Bellare-Rogaway model, this trusted third party should be accessed every time that a ciphertext is generated. By contrast, in our model the (rather different) trusted party is accessed only once, and, thereafter, registered users can generate ciphertexts on their own.

  (To be sure, the quite general implementation that we propose is not efficient, but this inefficiency is *not* due to our model.)

The Essence of our Implementation. Our scheme is based on those of [17, 18, 15], and makes use of the following key registration process: a user simply gives a zero-knowledge proof of knowledge of his secret key. Since the proof system is zero-knowledge, no registration authority (honest or dishonest) gains information about a user's secret key.

However, if a message m sent by user U to user V were only encrypted in V's public key, the fact that V properly registered his key guarantees that he will understand m, but the fact that U properly registered his own key offers no guarantee that he understands what he sent. Therefore, following [17], we make the encryption of a message depend on the public key of both sender and recipient. More precisely, giving a self-referential twist to the schemes of [15] and [18], our sender U encrypts a message for V both in V's two public keys and his own public key —and provides a proof of having done so.

We note, however, that as simple as our twist may be, it adds considerable complexity to our proof, even taking the schemes of [17] and [18] for granted. Indeed, even defining the ("fall-back") notion of chosen-ciphertext security becomes more complex in our new setting.[1]

We also note that our scheme is different from traditional public-key encryption in that it is not anonymous: unless substantial complications are added, a ciphertext reveals the public key of its sender (if the sender is honest). Nonetheless, anonymity is not a requirement for public-key encryption, and most public-key protocols do not even envisage it.

The Essence of Our Application. We apply plaintext awareness to the *Dolev-Yao* model[8], the famous alternative for cryptographic protocol analysis.[2] Unlike the more general computational models, the Dolev-Yao model has the advantage of extreme simplicity and ease of use. Although the general problem of

[1] **Warning**: At first glance, it would seems that both our model and our implementation could be greatly simplified. However, while this may indeed be the case, many "simplifications" (See Appendix) lead to intractable complications later on.

[2] It is also known as the *formal* model, due to its origins in the formal methods community.

determining correctness remains undecidable, the correctness of an impressive number of protocols has been successfully decided, even by *automated* tools [14, 16, 21].

However, these successes are qualified by their reliance on *extremely* strong assumptions. In particular, the Dolev-Yao model assumes that the adversary is not allowed to perform arbitrary computations. Instead, he is limited to selecting his actions from a small number of predetermined operations. For example, he is prohibited from doing anything with a ciphertext except decrypting it with the right key.

These restrictions raise serious doubts about the meaningfulness of the Dolev-Yao model. After all, a real-world adversary is not required to obey them. However, we show that plaintext awareness ensures that the Dolev-Yao restrictions can be actually *enforced* in the real world. It is here that the naturalness of our model and implementation matters crucially: were our model in any way abstract or unachievable, we would simply be reducing one abstraction to another. However, since our model is concrete, we show that the Dolev-Yao adversary can be made concrete also.

2. Preliminaries

2.1. **Enhanced GMR Notation.** For handling more complex probabilistic experiments, in our paper we need to **enhance** the standard GMR notation [12].

Standard GMR Notation.

- *Random assignments.* If S is a probability space, then "$x \stackrel{R}{\leftarrow} S$" denotes the algorithm which assigns to x an element randomly selected according to S. If F is a finite set, then the notation "$x \stackrel{R}{\leftarrow} F$" denotes the algorithm which assigns to x an element selected according to the probability space whose sample space is F and uniform probability distribution on the sample points.
- *Probabilistic experiments.* If $p(\cdot,\cdot,\cdots)$ is a predicate, the notation $\Pr[x \stackrel{R}{\leftarrow} S; y \stackrel{R}{\leftarrow} T; ... : p(x,y,\cdots)]$ denotes the probability that $p(x,y,\cdots)$ will be true after the ordered execution of the algorithms $x \stackrel{R}{\leftarrow} S$, $y \stackrel{R}{\leftarrow} T, ...$
- *Negligible Functions* We denote by $\nu : \mathcal{N} \rightarrow (0,1)$ a function that vanishes faster than the inverse of any fixed polynomial, for all sufficiently large arguments.

New GMR Notation.

- *History-Preserving Algorithms.* We say that an algorithm (or interactive TM) A is *history-preserving* (HP, for short) if it "never forgets" anything. As soon as it flips a coin or receives an input or a message, A writes it on a separate history tape that is write-only and whose head always moves from left to right. The history tape's content coincides with A's internal configuration before A executes any step.

  If A is an HP algorithm, then if A appears more than once in a piece of GMR notation (e.g, $\Pr[\ldots; a \stackrel{R}{\leftarrow} A(x); \ldots; b \stackrel{R}{\leftarrow} A(y); ... : p(\cdots, a, b, \cdots)]$) then the history and state of A is preserved from the end of one "use" to the beginning of the next.

  The notation $h \stackrel{H}{\longleftarrow} A$ indicates that h is the content of the current history tape of A.
- *Adversaries.* An *adversary* is an efficient history-preserving algorithm (interactive TM).
- *Non-Determinism.* If S is a set, the notation $x \stackrel{ND}{\longleftarrow} S$ indicates that x has been non-deterministically chosen from S.

  Whenever non-deterministic choices appear within an experiment, they are regarded as constants (and not random variables), and all probabilistic statements made refers to each possible individual choice. For instance, the expression $Pr(x \stackrel{ND}{\longleftarrow} S : A(x) = 1) = 1/2$ means that, for every $x \in S$, A on x outputs 1 with exactly probability $1/2$.

2.2. **Protocols.** Following [11], we consider a two-party protocol as a pair, (A, B), of interactive Turing machines. Briefly, on input (x, y), where x is a private input for A and y a private input for B, and random input (r_A, r_B), where r_A is a private random tape for A and r_B a private random tape for B, protocol (A, B) computes in a sequence of rounds, alternating between A-rounds and B-rounds. In an A-round (B-round) only A (only B) is active and sends a message (i.e., a string) that will become an available input to B (to

A) in the next B-round (A-round). A computation of (A, B) ends in a B-round in which B sends the empty message and computes a private output.[3]

Executions, Transcripts, and Outputs. If (A, B) is a protocol and (x, y) an input for (A, B), we let $EXE^{A,B}(x|y)$ denote the experiment of randomly executing (A, B) on input (x, y). In our definitions, we think of this experiment as affecting the computation history of the participants rather than having an output.

Letting E be an execution of protocol (A, B) on input (x, y) and random input (r_A, r_B), we make the following definitions:

- The *transcript* of E consists of the sequence of messages exchanged by A and B, and is denoted by $TRANS^{A,B}(x, r_A|y, r_B)$;
- The *output of B in E* consists of the string z output by B in the last round of E, and is denoted by $OUT_B^{A,B}(x, r_A|y, r_B)$, though it only depends on B's inputs, coin tosses, and E's transcript.

We also define the following random distributions

- $TRANS^{A,B}(x, \cdot|y, r_B)$, $TRANS^{A,B}(x, r_A|y, \cdot)$, and $TRANS^{A,B}(x, \cdot|y, \cdot)$

as the distribution respectively obtained by randomly selecting r_A, r_B, or both, and then drawing from $TRANS^{A,B}(x, r_A|y, r_B)$. We also consider the similarly defined random variables

- $OUT_B^{A,B}(x, \cdot|y, r_B)$, $OUT_B^{A,B}(x, r_A|y, \cdot)$, and $OUT_B^{A,B}(x, \cdot|y, \cdot)$;

In all above quantities the superscript (A, B) will sometimes be omitted when clear from the context. When we do not wish to explicitly consider the random coins of the participants, we will omit them.

Security parameters. We say that an execution of a protocol (A, B) has *security parameter k* if the private input of A is of the form $(1^k, x)$ and the private input of B is of the form $(1^k, y)$. (Thus, *de facto* 1^k is a "common input" while x and y are the "real private inputs.")

3. The Notion of Plaintext Awareness via Key Registration

3.1. **Informally.** Let us try to summarize our notion in a way that is as independent as possible from our chosen implementation.

In essence, a plaintext aware cryptosystem with key registration consists of two different public-key encryption schemes that share the same encryption algorithm ($(\mathsf{G}_1, \mathsf{E}, \mathsf{D}_1)$ and $(\mathsf{G}_2, \mathsf{E}, \mathsf{D}_2)$) together with a key-registration protocol $(\mathsf{RU}, \mathsf{RA})$.

Each user has two different roles, sender and recipient, and has a different encryption/decryption key pair for each role. A user U runs G_1 to generate his own recipient keys (e_r, d_r) and G_2 to generate his own sender keys (e_s, d_s). Collectively, we may view e_s and e_r as a single public key $e = (e_s, e_r)$ and ds and dr as a single private key $d = (d_s, d_r)$. Only e's public sender portion (e_s) must be registered in a special way.

The registration of e_s is performed by having U run protocol RU on private input (e_s, d_s) with the registration authority running RA on the empty input. A successful registration is signified by having the registration authority simply outputting the key e_s. The key will then be inserted in the public file, or a certificate for e_s will be handed out to U. The precise mechanism of the publication of e_s is irrelevant here. What is crucial, however, is that the registration protocol be a *'secure atomic operation.* That is, we can think of it as being run one user at a time, in person, and from beginning to end. (Without this assumption, we would have to worry about man-in-the-middle, concurrency and other types of attacks which will obscure both the definitional and implementation aspects of our model.)

Encrypted messages can be successfully exchanged only among users with registered public keys. When user U wishes to send an encrypted message m to user V, he runs algorithm E on inputs m, his sending public key, and V's public receiving key. The so generated ciphertext c includes a tag specifying the sender and recipient (i.e., the public keys of the sender and the recipient).

User V decrypts a c by running the recipient decryption algorithm D_1 on inputs c, his own private receiving key, and U's public sending key e_s. This operation returns either an alleged plaintext, or the distinguished symbol $\perp$ to indicate an invalid ciphertext. Should e_s key not be properly registered, we assume that the return value will always be $\perp$.

[3]Due to the one-sidedness of secure computation, only machine B produces an output.

In this setting, plaintext awareness has the following meaning: If the adversary registers a key with the proper registration authority, then the adversary can decrypt *any* ciphertext it addresses to an honest user exactly as that user will decrypt it. (We note that the special symbol $\perp$ is a possible valid decryption here, but this possibility does not arise between honest users. Thus decrypting all cipher texts with $\perp$ will not suffice!)

Of course, an adversary may not comply with anyone's requests, including those to decrypt the ciphertexts he makes. So, when we say that the adversary can decrypt its ciphertexts just as the honest participants would, we actually mean that the output of an honest recipient is also easily extracted from the adversary's computational state after it finishes generating the ciphertext. (Recall that the adversary is defined to be a history preserving algorithm, and thus this history encompasses the adversary's state during key registration.) This is a natural extension of Bellare and Rogaway's notion. In their case, in fact, the plaintext was retrievable from a smaller portion of his own computational history: just the messages the adversary made to the random oracle. We ensure, however, that when trying to output a ciphertext he does not understand the adversary has all the help he can meaningfully get. In particular, we assume that he has overheard the registration process of the honest recipient.

Because users engage in key-registration protocols with the registration authority in order to gain plaintext awareness, it is crucial that this process never degrade the security of the underlying encryption. That is, we require that the scheme be secure against adaptive chosen-ciphertext attack (i.e., as secure as possible without any central authority) even if the registration authority were to be untrustworthy.

To simplify the formalization below (as well as any implementation), we simply assume that the registrant has "veto power" over his own portion of the public file. That is, no key other than a user's true public key e will be registered under his name. (If we do not assume this, much more complex key registration protocols would be needed.) This simplification allows us to simply our formal properties to be about honestly generated keys rather than successfully registered keys.

3.2. **More Formally.** A *registration-based plaintext-aware encryption scheme* is a sextuple $(\mathsf{G}_1, \mathsf{G}_2, \mathsf{E}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{RU}, \mathsf{RA})$, where[4]:

- $\mathsf{G}_1 : \mathsf{Parameter} \to \mathsf{PublicKey} \times \mathsf{PrivateKey}$
  is a probabilistic and efficient algorithm, called the *receiver key generation algorithm*,
- $\mathsf{G}_2 : \mathsf{Parameter} \to \mathsf{PublicKey} \times \mathsf{PrivateKey}$
  is a probabilistic and efficient algorithm, called the *sender key generation algorithm*,
- $\mathsf{E} : \mathsf{String} \times \mathsf{PublicKey} \times \mathsf{PublicKey} \to \mathsf{Ciphertext}$
  is a probabilistic and efficient algorithm, called the *encryption algorithm*,
- $\mathsf{D}_1 : \mathsf{String} \times \mathsf{PrivateKey} \times \mathsf{PublicKey} \to \mathsf{String} \cup \{\perp\}$
  is an efficient algorithm, called the *recipient decryption algorithm*,
- $\mathsf{D}_2 : \mathsf{String} \times \mathsf{PrivateKey} \times \mathsf{PublicKey} \to \mathsf{String} \cup \{\perp\}$
  is an efficient algorithm, called the *sender decryption algorithm*, and
- $(\mathsf{RU}, \mathsf{RA})$
  is a two-party protocol, called *key-registration,*

which satisfies the following conditions:

- *Honest Registration*: The key registration protocol between an honest registrant and an honest registration authority will always be successful:

$$
\begin{array}{ll}
\forall k \in \mathsf{Parameter} & \\
\Pr[& (d,e) \xleftarrow{R} \mathsf{G}_1(1^k); \\
 & o \xleftarrow{R} OUT_{\mathsf{RA}}^{\mathsf{RU},\mathsf{RA}}((e,d), \cdot | \varepsilon, \cdot) : \\
 & o = e_1 \qquad\qquad\qquad\qquad] = 1
\end{array}
$$

[4]The helper sets are:

- $\mathsf{Parameter} = \mathcal{N}$, the natural numbers,
- $\mathsf{Coins} = \{0,1\}^{\omega}$, the set of all infinite sequences of bit-strings.
- $\mathsf{PublicKey}$, $\mathsf{PrivateKey}$ and $\mathsf{Ciphertext}$ vary between key generation algorithms and implicitly depend on the parameter, and
- $\mathsf{String} = \{0,1\}^*$

- *Dishonest Registration*: The registrant exercises veto power over the registration of his own keys:

$$\begin{array}{ll} \forall \mathsf{A},\ \forall k \in \mathsf{Parameter} & \\ \Pr[\quad (d,e) \stackrel{R}{\leftarrow} \mathsf{G}_1(1^k); & \\ \quad o \stackrel{R}{\leftarrow} OUT_{\mathsf{A}}^{\mathsf{RU},\mathsf{A}}((e,d),\cdot|\varepsilon,\cdot): & \\ \quad o = e \vee o = \perp &] = 1 \end{array}$$

- *Honest Symmetry*: An honest sender will "always" understand the same thing as an honest recipient:

$$\begin{array}{ll} \forall c \in \mathsf{String} & \\ \Pr[\quad (d_1,e_1) \stackrel{R}{\leftarrow} \mathsf{G}_1(1^k);\ (d_2,e_2) \stackrel{R}{\leftarrow} \mathsf{G}_2(1^k): & \\ \quad \mathsf{D}_1(c,d_1,e_2) = \mathsf{D}_2(c,d_2,e_1) &] \geq 1 - \nu(k) \end{array}$$

- *Honest Completeness*: An honest sender will always be understood by an honest recipient:

$$\begin{array}{ll} \forall m \in \mathsf{Plaintext} & \\ \Pr[\quad (d_1,e_1) \stackrel{R}{\leftarrow} \mathsf{G}_1(1^k);\ (d_2,e_2) \stackrel{R}{\leftarrow} \mathsf{G}_2(1^k); & \\ \quad c \stackrel{R}{\leftarrow} \mathsf{E}_2(m,e_1,e_2); & \\ \quad g \stackrel{R}{\leftarrow} \mathsf{D}_1(c,d_1,e_2): & \\ \quad g = m &] = 1 \end{array}$$

- *Honest Security:* If recipient and sender are honest, their channel is chosen-ciphertext secure

$$\begin{array}{ll} \forall \text{ oracle-calling adversaries } \mathsf{A}, & \\ \Pr[\quad (d_1,e_1) \stackrel{R}{\leftarrow} \mathsf{G}_1(1^k);\ EXE^{\mathsf{RU},\mathsf{A}}((d_1,e_1),\cdot|\varepsilon,\cdot); & \\ \quad (d_2,e_2) \stackrel{R}{\leftarrow} \mathsf{G}_2(1^k);\ EXE^{\mathsf{RU},\mathsf{A}}((d_2,e_2),\cdot|\varepsilon,\cdot); & \\ \quad m_0,m_1 \stackrel{R}{\leftarrow} \mathsf{A}^{D_1(\cdot,d_1,e_2)}(e_1,e_2); & \\ \quad b \stackrel{R}{\leftarrow} \{0,1\}; & \\ \quad c \stackrel{R}{\leftarrow} \mathsf{E}_1(m_b,e_1,e_2); & \\ \quad g \stackrel{R}{\leftarrow} \mathsf{A}^{D_1(\cdot,d_1,e_2)-\{c\}}(c): & \\ \quad b = g &] \geq \frac{1}{2} - \nu(k) \end{array}$$

  where
  - m_0 and m_1 have the same length,
  - $D_1(\cdot,d_1,e_2) - \{c\}$ is the oracle that, on input c', returns $\mathsf{D}_1(c',d_1,e_2)$ as long as $c' \neq c$.
- *Plaintext Awareness:* If registration authority is honest and the adversary is a properly registered user, then the adversary should, for any string he sends to a properly registered recipient, know in advance how the recipient will decrypt it:

$$\begin{array}{ll} \exists \text{ efficient algorithm } \mathsf{S},\ \forall \text{ adversaries } \mathsf{A}, & \\ \Pr[\quad (e_1,d_1) \stackrel{R}{\leftarrow} \mathsf{G}(1^k); & \\ \quad t \stackrel{R}{\leftarrow} TRANS^{\mathsf{RU},\mathsf{RA}}((d_1,e_1),\cdot|\varepsilon,\cdot); & \\ \quad key \stackrel{R}{\leftarrow} OUT_{\mathsf{RA}}^{\mathsf{A},\mathsf{RA}}((t,e_1),\cdot|\varepsilon,\cdot); & \\ \quad h \stackrel{H}{\longleftarrow} \mathsf{A}; & \\ \quad c \stackrel{ND}{\longleftarrow} \{0,1\}^k: & \\ \quad \mathsf{S}(\mathsf{A},h,e_1,key,c) = \mathsf{D}(key,d_1,c) &] \geq 1 - \nu(k) \end{array}$$

(Note the use of the history-revealing $\stackrel{H}{\longleftarrow}$ operator, and the non-deterministic $\stackrel{ND}{\longleftarrow}$ operator.)

4. Implementing Plaintext Awareness via Key Registration

Our scheme uses non-interactive zero-knowledge proofs [5, 20, 6] in order to enhance encryption security. This approach has been pioneered by Naor and Yung [15], and greatly refined by Sahai [18]. Another fount of inspiration comes from the work of Rackoff and Simon [17], that used a very powerful registration authority (indeed one that chooses every user's secret keys) to obtain chosen-ciphertext security.

We make use the following three cryptographic tools:

- $(\mathsf{G}',\mathsf{E}',\mathsf{D}')$, a semantically secure cryptosystem in the sense of [10], where G' is the key generation algorithm, E' is the encryption algorithm, and D' is the decryption algorithm.

- (f, P, V, S), a non-malleable NIZK proof system for NP, in the sense of [18], where P is the proving algorithm, V is the verification algorithm, S is the simulator, and $f(k)$ is the length of the reference string for security parameter k.
- a zero-knowledge proof of knowledge for NP [3, 9, 11].

The underlying complexity assumption of our scheme is the existence of trapdoor permutations, (which suffices for non-malleable NIZK proof system for NP, for ZK proof of knowledge for NP, and for semantically secure public-key encryption).

4.1. **The Scheme $\mathcal{S}$.** The scheme $\mathcal{S} = (\mathcal{G}_1, \mathcal{G}_1, \mathcal{E}, \mathcal{D}_1, \mathcal{D}_1, \mathcal{RU}, \mathcal{RA})$ is as follows. Key generation is split into two parts, as each entity need not act as both a receiver and a sender. To be semantically in line with the definitions presented earlier, we could simply have every entity run both the receiver key generation algorithm and the sender key generation algorithm, but this will make notation later unnecessarily burdened.[5]

- $\mathcal{G}_1$ (receiver key generation): Generate (e_1, d_1) and (e_2, d_2) according to $\mathsf{G}'(1^k)$. Pick a random σ from $\{0,1\}^{f(k)}$. The public (receiver's) key is (e_1, e_2, σ) and the secret key is (d_1, d_2).
- $\mathcal{G}_2$ (sender key generation): Generate (e_3, d_3) according to $\mathsf{G}'(1^k)$. The public (sender's) key is e_3 while the secret key is d_3.
- $\mathcal{RU}$ and $\mathcal{RA}$, on input e_3, engage in a zero-knowledge proof of knowledge that the user knows d_3. If the zero-knowledge proof of knowledge terminates correctly, $\mathcal{RA}$ outputs the sender's public key, otherwise it outputs $\perp$. Note that in this key registration protocol, the receiver's keys are ignored.
- $\mathcal{E}$, on input $(m, (e_1, e_2, \sigma), (e_3))$ first computes $c_1 = \mathsf{E}'(e_1, m)$, $c_2 = \mathsf{E}'(e_2, m)$, and $c_3 = \mathsf{E}'(e_3, m)$. Here, naturally, e_1, e_2, and σ are from the receiver's public key, while e_3 is the sender's public key. Then, it computes π, a non-malleable NIZK proof that c_1, c_2, and c_3 all encrypt the same message relative to e_1, e_2, and e_3, respectively. It outputs (c_1, c_2, c_3, π).
- $\mathcal{D}_1$, on input $((c_1, c_2, c_3, \pi), (e_1, e_2, \sigma, d_1, d_2), (e_3))$ checks if π is a valid proof that c_1, c_2 and c_3 are encryptions of the same messages under e_1, e_2 and e_3, respectively, relative to the reference string σ. If so, it outputs $\mathsf{D}(d_1, c_1)$. Otherwise, it outputs $\perp$.
- $\mathcal{D}$, on input $((c_1, c_2, c_3, \pi), (e_3, d_3), (e_1, e_2, \sigma)$ checks if π is a valid proof that c_1, c_2 and c_3 are encryptions of the same messages under e_1, e_2 and e_3, respectively, relative to the reference string σ. If so, it outputs $\mathsf{D}(d_3, c_3)$. Otherwise, it output $\perp$.

4.2. **Security of $\mathcal{S}$.**

4.2.1. *$\mathcal{S}$ Enjoys Honest Registration.* This is natural: the registration process is a zero-knowledge protocol. By its completeness property, an honest prover will always be able to prove a true theorem x to an honest verifier if it possesses a witness w. In this case, the theorem is e_1 and the witness is d_1. Since the honest registrant has access to the witness and engages in the protocol honestly, the honest registration authority will always accept the proof and output the public key.

4.2.2. *$\mathcal{S}$ Enjoys Dishonest Registration.* We assume that this is ensured by some external mechanism.

4.2.3. *$\mathcal{S}$ Enjoys Honest Symmetry.* If (c_1, c_2, c_3, π) is received by the honest recipient or sender and π is rejected by the verifier V, then both the recipient D_1 and the sender D_2 will output $\perp$. If, however, the proof π is accepted by the verifier, then by the soundness of the proof system we know that c_1, c_2 and c_3 will (almost) always contain the same plaintext. Hence, when D_1 decrypts c_1 it will almost always produce the same output as when D_2 decrypts c_3. (And if the input is not of the form (c_1, c_2, c_3, π), then both D_1 and D_2 produce $\perp$.) Hence, for any input, both D_1 and D_2 will almost always produce the same output.

4.2.4. *$\mathcal{S}$ Enjoys Honest Completeness.* This should be clear. If the sender is honest, then it produces (c_1, c_2, c_3, π) where c_1, c_2 and c_3 all contain the same plaintext m and π is an honest proof of that fact. Since the proof is honest, the recipient will accept it and decrypt c_1 to receive m, the same message encrypted by the sender.

[5] In these definitions, we liberally assume that any secret key contains any needed information from the associated public key.

4.2.5. *S Enjoys Honest Security.* We will prove chosen-ciphertext security by the contrapositive. Suppose there is an adversary A that succeeds in an adaptive chosen ciphertext attack against an honest sender and an honest recipient. We will give two reductions, R and R', and we will prove that one of the two must break the underlying encryption scheme

The basic structure of R is that we simulate the adaptive chosen-ciphertext attack. R attempts to break the security of the underlying cryptosystem. It must, on input e, return a pair of messages, and then on input c (a ciphertext, the encryption of one of the two messages), decide which message c encrypts with probability non-negligibly greater than 1/2.

On input $e, 1^k$, R runs as follows.

1. First, we create the receiver's public key (e_1, e_2) and the sender's public key (e_3) as follows. Pick a at random from $\{1,2\}$. Set e_{3-a} to be e and set $(e_a, d_a) \stackrel{R}{\leftarrow} \mathsf{G}'(1^k)$. Generate σ according to the simulator S for the NIZK proof system[6]. Set $(e_3, d_3) \stackrel{R}{\leftarrow} \mathsf{G}'(1^k)$.
2. Run A on input $((e_1, e_2, \sigma), (e_3))$. Whenever A asks for a decryption of (c_1', c_2', c_3', π'), we check the correctness of π' using V. If it verifies, we decrypt c_a' using d_a and output that as the result. Otherwise we return $\perp$.
3. Eventually A will output (m_0, m_1) Output (m_0, m_1) and obtain challenge c. For notation later, let us say that m_β is the message c encrypts.
4. We then simulate the ciphertext challenge for A. Pick b at random from $\{0,1\}$. Let $c_a \stackrel{R}{\leftarrow} \mathsf{E}'(e_a, m_b)$, let $c_{3-a} \stackrel{R}{\leftarrow} c$. With probability 1/2, let $c_3 \stackrel{R}{\leftarrow} \mathsf{E}'(e_3, m_b)$ and otherwise, let $c_3 \stackrel{R}{\leftarrow} \mathsf{E}'(e_3, m_{1-b})$.[7] Fake the NIZK proof π using the simulator S (with auxilliary information τ).
5. Run A on input (c_1, c_2, c_3, π).
6. Again, whenever A asks for a decryption, we check the proof and decrypt using d_a.
7. Eventually A outputs an answer b'. If $b = b'$, output b'. Otherwise, output a random bit.

There are three kinds of input the adversary can get.

I. First, it is possible that c_1, c_2, and c_3 all encrypt the same message m_β. In this case, the input given to the adversary is indistinguishable from the input in the real attack the adversary succeeds in. Thus, the adversary must return β with probability $1/2 + \epsilon$, where ϵ is some non-negligible function of k.

II. Second, it may be that c_1 and c_2 both encrypt the same message m_β but c_3 encrypts $m_{1-\beta}$. Let us say that in this case, the adversary returns β with probability x.

III. Finally, it may be that c_1 and c_2 encrypt different messages, and that c_3 encrypts the message m'_β. Let us say that in this case the adversary returns β' with probability y.

   Note that there are two subcases:
   - c_a and c_3 encrypt the same message while c_{3-a} encrypts the other, and
   - c_{3-a} and c_3 encrypt the same message while c_a encrypts the other

   These two cases are indistinguishable to the adversary. Since the adversary cannot make any proofs of false theorems with non-negligible probability, if the adversary ever makes a decryption query when c_1 and c_2 encrypt different messages, the answer will be $\perp$. Thus, the case $a = 1$ and the case $a = 2$ give the same distribution. (See [18]. This is an encapsulated version of one of the main details from Sahai's proof that his scheme is CCA2-secure.)

This reduction is parameterized by the values x and y, both of which can be chosen by the adversary. However, we will show that the only value of interest to us is x. In fact, we will show that for almost all values of x, the above reduction breaks the security of $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$. However, the reduction R will not work for certain values of x, so we give a different reduction R' and show that it does.

To begin: what is the probability that R returns the correct answer? Again, we consider two cases: when $b = \beta$ and when $b \neq \beta$:

- In the case that $b = \beta$, the adversary sees an input of type I with probability 1/2. When the adversary sees an input of type I, R is correct with probability $3/4 + \epsilon/2$. If the adversary does not see an input of type I though $b = \beta$ then it sees an input of type II, in which case R is correct with probability $x + (1 - x)/2$ (since whenever the adversary returns β in an input of type 2, R is correct, and the

[6] Here, we assume the simulator S is history-preserving.

[7] An interesting case arises when the probability of encrypting m_b is generalized from 1/2 to p. See footnote 8.

rest of the time, R is correct with probability $1/2$). Thus, the total probability that R is correct when $b = \beta$ is $\frac{1}{2}((3/4+\epsilon/2)+(1/2+x/2))$.

- Now let us examine the case that $b \neq \beta$. Any time this is true, we give input type III to the adversary. However, with probability $1/2$, m_b encrypted into c_3 and with probability $1/2$, m_{1-b} is encrypted into c_3. (Recall, these two cases are indistinguishable to the adversary.) Thus, the adversary returns b with probability $\frac{1}{2}y + \frac{1}{2}(1-y)$, and otherwise returns $1-b$. In this case, our total probability of being correct is $(y/4)+(1-y)/4$, since we are only correct when the adversary returns $1-b$, and then, only half the time.

Taking into account all cases, the probability that R is correct is

$$\frac{1}{2}\left(\frac{1}{2}(3/4+\epsilon/2)+\frac{1}{2}(1/2+x/2)+\frac{1}{2}(1/2-y/2)+\frac{1}{2}(y/2)\right)$$

This expression evaluates to

$$\frac{3}{16}+\frac{\epsilon}{8}+\frac{1}{8}+\frac{x}{8}+\frac{1}{8}=\frac{7}{16}+\frac{\epsilon+x}{8}.$$

Now if $\epsilon+x$ is non-negligibly different from $1/2$ then the above expression is also, and R breaks $(\mathsf{G}',\mathsf{E}',\mathsf{D}')$.[8]. If, on the other hand, $x \approx 1/2-\epsilon$, then we can use A to break the security of $(\mathsf{G}',\mathsf{E}',\mathsf{D}')$ directly. Let R' be the reduction:

1. Generate (e_1,d_1) and (e_2,d_2) by running $\mathsf{G}'(1^k)$. Generate σ and τ (any auxilliary information) according to the simulator S for the NIZK proof system. Set $e_3 \stackrel{R}{\leftarrow} e$.
2. Run A on input $((e_1,e_2,\sigma),(e_3))$. Whenever A asks for a decryption query we check the correctness of the included NIZK proof using V. If it verifies, we decrypt c_1 using d_1 and output that as the result, otherwise we return $\perp$.
3. Obtain m_0, m_1 as the output of A. Output (m_0,m_1) and obtain challenge c. For notation later, let us say that m_β is the message c encrypts.
4. Pick b at random from $\{0,1\}$. Let $c_1 \stackrel{R}{\leftarrow} \mathsf{E}'(e_1,m_b)$, let $c_2 \stackrel{R}{\leftarrow} \mathsf{E}'(e_2,m_b)$, and let c_3 be c.
5. Fake the NIZK proof π using the simulator S (with auxilliary information τ).
6. Run A on input (c_1,c_2,c_3,π). Again, whenever A asks for a decryption, we check the proof and decrypt using d_1. Eventually A outputs an answer b'. Output b'.

The proof that R' works is simple. If R' picks $b=\beta$ then A outputs b with probability $1/2+\epsilon$. If R' picks $b \neq \beta$ then A sees input type II and so it outputs b with probability $x = 1/2-\epsilon+\nu'$ where ν' is (positively or negatively) negligible. Thus in either case, we output β with probability at least $1/2+\epsilon-|\nu'/2|$.

4.2.6. *$\mathcal{S}$ Enjoys Plaintext Awareness.* This is fairly simple, and we show it informally. The extractor works as follows. On input (A,h,e_1,key,c), we use H to rewind the adversary to the point where A engages in key registration with RA, and uses the extractor from the interactive zero knowledge proof of knowledge to find a value d. The extractor then uses d to decrypt c and give the answer. From the extractibility property of the proof system, d must be a secret key relative to e_1 so this answer is correct.

5. Plaintext Awareness and the Dolev-Yao Adversary

We conclude by considering a naturally-arising application of plaintext-awareness: the adversary of the Dolev-Yao model of cryptographic protocols [8].

The Dolev-Yao model is an alternate model of cryptographic protocol execution which grew out of the formal methods community. It differs from the standard, computational, model in two important ways:

1. The representation of messages, and
2. The ability of the adversary.

[8]If we had kept a general p through the entire reduction, we could have also said interesting things about y. Letting $p=1$ instead of $1/2$ shows that $y-\epsilon$ must be negligibly close to $1/2$. What's more, if $x \approx 1/2-\epsilon$ and $y \approx 1/2+\epsilon$ then the probability that R is correct is negligibly close to $1/2$, *regardless of p*.

In this model, messages are not bit-strings but elements of some algebra $\mathcal{A}$. In essence, messages *are* parse trees in which the leaves are names ($\mathcal{M}$), random numbers ($\mathcal{R}$) or keys ($\mathcal{K}_{Pub}$ and $\mathcal{K}_{Priv}$).[9] Compound messages are formed using two operations: Compound messages are created by two operations:

- *encrypt* : $\mathcal{K}_{Pub} \times \mathcal{A} \to \mathcal{A}$
- *pair* : $\mathcal{A} \times \mathcal{A} \to \mathcal{A}$

Because messages are parse trees, every message has a unique interpretation. We assume for our purposes that the algebra has contains a finite number of atomic elements, though the model itself has no such restriction.

Whereas the standard adversary is an arbitrary algorithm, the Dolev-Yao adversary is much more limited. As in the standard model, the adversary is able to know all public and predictable values. Likewise, the adversary controls the network in both models, meaning that it sees and routes all traffic between honest participants. However, when it comes to the ability of the adversary to create new messages, the two models sharply differ.

Where the standard adversary is able to create any efficiently computable bit-string, the Dolev-Yao model can only create new parse trees by applying to known ones a limited number of operations: pairing, separation of pairs, encryption in public keys, and decryption in known keys. Formally, the power of the Dolev-Yao adversary to create new messages is given by a set-theoretic operation:

Definition 1 (Closure)**.** The *closure of* S, written $C[S]$, is the smallest set such that:

1. $S \subseteq C[S]$,
2. If $\{\!|M|\!\}_K \in C[S]$ and $K^{-1} \in C[S]$, then $M \in C[S]$,
3. If $M \in C[S]$ and $K \in C[S]$, then $\{\!|M|\!\}_K \in C[S]$,
4. If $M\,N \in C[S]$, then $M \in C[S]$ and $N \in C[S]$, and
5. If $M \in C[S]$ and $N \in C[S]$, then $M\,N \in C[S]$.

(It is assumed that S contains all public values such as names and public keys.)

It is the central assumption of the Dolev-Yao model that the closure operation is the extent of the adversary's ability to manipulate cryptographic material:

Definition 2 (Formal Adversary)**.** The formal adversary is a non-deterministic process on $\mathcal{A}$ that, given a set S of messages, produces messages in $C[S]$.

The Dolev-Yao is an attractive model in which to work. Proofs are simple and easily found, and protocol verification is easily automated. However, it is not clear how the Dolev-Yao model relates to the standard computational model. The above restriction makes the formal adversary seem fairly weak: the standard adversary can certainly calculate any value available to the formal adversary, and many more besides. Hence, security against the formal adversary seems like a fairly weak property. However, it turns out that if the underlying cryptography is plaintext aware, then the standard adversary is no more powerful than the formal adversary. That is, computational cryptography can limit the computational adversary to this closure operation.

To formalize the limitation on the formal adversary in terms of computational cryptography, we need to somehow translate the parse-tree messages of the Dolev-Yao model into bit-strings. To do this, we adapt the "encoding" operation from Abadi and Rogaway [1] from the symmetric-encryption setting to that of asymmetric encryption. In brief, the "encoding" of a message M, written $[\![M]\!]_n$, depends on the parse tree of M, the security parameter, and the choice of underlying public-key encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$.[10] Recursing on the structure of M:

- If M is the nonce of an honest participant, then $[\![M]\!]_n$ is a specific n-bit string, chosen at random.
- If M is a nonce of the formal adversary, then $[\![M]\!]_n$ is a specific n-bit string chosen by the computational adversary
- If M is a public or private key of an honest participant, then $[\![M]\!]_n$ is a specific computational key chosen at random from $G(1^n)$.
- If M is a public or private key of the formal adversary, then $[\![M]\!]_n$ is a specific computational key chosen by the computational adversary

[9] We will only consider the case of asymmetric encryption, though the Dolev-Yao models symmetric encryption also.
[10] Our definition of plaintext-aware encryption contains several more algorithms, which we ignore for the moment.

- If $M = M_1\, M_2$, then $[\![M]\!]_n$ is the concatenation of $[\![M_1]\!]_n$ and $[\![M_2]\!]_n$.
- If $M = \{\!|M_1|\!\}_K$, then $[\![M]\!]_n$ is the *distribution* on bit-strings defined by $\mathsf{E}([\![M_1]\!]_n, [\![K]\!]_n)$.

Now that we can relate Dolev-Yao messages and bit-strings, we can formalize the intuition of Definition 2. We will call a computational encryption scheme *ideal* if it restricts the computational adversary to the limit on the Dolev-Yao adversary:

Attempt 1. An encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is *ideal* if

$$\begin{array}{ll} \multicolumn{2}{l}{\forall\, \mathsf{A}_{PPT},\ \forall\, S \subseteq \mathcal{A},\ \forall\, M \notin C[S],\ \forall \text{polynomials } q,\ \forall \text{ sufficiently large } n:} \\ \Pr[& s \stackrel{R}{\leftarrow} [\![S \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Subv} \cup \mathcal{M}]\!]_n\, ; \\ & m \stackrel{R}{\leftarrow} \mathsf{A}(1^n, s) : \\ & m \in \mathbf{supp}\, [\![M]\!]_n \qquad\qquad] \leq \frac{1}{q(n)} \end{array}$$

(Here, $\mathbf{supp}(D)$ means the support of distribution D.)

However, our results are subject to one technical limitation: S must be *acyclic*. A formal definition of an acyclic set can be found in [13]. Informally, it means that if K_1 encrypts K_2^{-1} in S K_2 encrypts K_3^{-1}, and so on, this sequence never loops back on itself.[11].

Hence, we revise the security condition:

Definition 3. An encryption scheme $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ is *ideal* if the adversary cannot create something outside the closure:

$$\begin{array}{ll} \multicolumn{2}{l}{\forall\, \mathsf{A}_{PPT},\ \forall \text{ acyclic } S \subseteq \mathcal{A},\ \forall\, M \notin C[S],\ \forall \text{polynomials } q,\ \forall \text{ sufficiently large } n:} \\ \Pr[& s \stackrel{R}{\leftarrow} [\![S \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Subv} \cup \mathcal{M}]\!]_n\, ; \\ & m \stackrel{R}{\leftarrow} \mathsf{A}(1^n, s) : \\ & m \in \mathbf{supp}\, [\![M]\!]_n \qquad\qquad] \leq \frac{1}{q(n)} \end{array}$$

This definition, it turns out, is no stronger than a strengthened form of plaintext-awareness. In particular, we need *authenticated* plaintext-awareness:

Definition 1. An *authenticated* plaintext-aware encryption scheme is a scheme $(\mathsf{G}_1, \mathsf{G}_2, \mathsf{E}, \mathsf{D}_1, \mathsf{D}_2, \mathsf{RU}, \mathsf{RA})$, which is

1. registration-based plaintext-aware, and
2. authenticated, in that the adversary cannot forge a message that seems to be between two honest participants:

$$\begin{array}{ll} \multicolumn{2}{l}{\forall \mathsf{A}_{PPT}\, \forall \text{ s.l. } k} \\ \Pr[& (e_1, d_1) \stackrel{R}{\leftarrow} \mathsf{G}(1^k); \\ & EXE^{\mathsf{RU},\mathsf{A}}((d_1, e_1), \cdot|\varepsilon, \cdot)\, ; \\ & (e_2, d_2) \stackrel{R}{\leftarrow} \mathsf{G}(1^k); \\ & EXE^{\mathsf{RU},\mathsf{A}}((d_2, e_2), \cdot|\varepsilon, \cdot)\, ; \\ & c \stackrel{R}{\leftarrow} \mathsf{A}^{D_1(\cdot, d_1, e_2) - \{c\}}(e_1, e_2); \\ & \mathsf{D}(e_1, d_2, c) = \perp \qquad\qquad] \geq 1 - \frac{1}{2^k} \end{array}$$

where $D_1(\cdot, d_1, e_2) - \{c\}$ is the oracle that, on input c', returns $\mathsf{D}_1(c', d_1, e_2)$ as long as $c' \neq c$.

We can achieve authenticated plaintext-aware encryption by adding public-key signatures to plaintext awareness. In our construction above, the sender generates a signature key-pair (in addition to the encryption key-pair) and proves knowledge of the signature generation key. Encryption proceeds as before, except that the sender now also signs the ciphertext[12]. The receiver now checks the signature in addition to the proof.

Theorem 4. Any encryption scheme that achieves authenticated plaintext-awareness is also ideal, if all public keys are registered with an honest RU.

Proof Sketch. Suppose that the encryption scheme were not ideal. Then with negligible probability the adversary could create an encoding m such that m is the valid encoding of an M not in $C[S]$.

Consider the parse tree of M. Each node in this tree is a message. Furthermore, if the adversary can create an encoding of an internal node of this tree with some probability p, then either

[11] This is a reasonable assumption for most "real-world" protocols, for reasons discussed in [13]

[12] Encryption now uses both of the sender's public and private keys, in addition to the recipient's public key, so that it has access to the sender's signature generation key.

1. That node is in $C[S]$, or
2. The adversary can, with probability almost p, create encodings of both children.

To see this, suppose the node is not in $C[S]$ and consider its type. It is easy to separate the components of a pair. On the other hand, if the adversary creates an encryption, there are two options. Keys are registered either by honest participants or the adversary. If the two keys used in the encryption were created honestly, then the chances that the adversary can create a valid encryption is negligible— a contradiction. If, on the other hand, the adversary created one key (and registered it) then the simulator allows us to turn that same adversary into a decryption oracle to get the plaintext. Hence, the adversary can create the encoding of an encryption, then run the simulator to extract the plaintext of the encryption. Since all public keys are known to the adversary, it can create both the encryption key and the plaintext, which are the two children.

Furthermore, membership in $C[S]$ is closed up the tree: if both children are in $C[S]$, then their parent is in $C[S]$ also. Hence, if M is not in $C[S]$ then there must be one path from root to leaf in the parse tree of M where no element of the path is in $C[S]$. (If there were no such path, then M would be in $C[S]$, a contradiction.)

By recursing down the tree and making both children of every node along this path, the adversary can create an encoding of the leaf at the end of this path. Hence, the adversary can make M, the root message, with probability p, then with probability $\frac{p}{q(n)}$ (for some polynomial q) the adversary can create the encoding of some atomic message M' outside of $C[S]$.

There are two cases:

1. If M' is related to S, then it must be either as the plaintext of an encryption or as the private key of some public key used in S (or both). In this case, the adversary has broken the security of the encryption. If we assume that every party has engaged in the setup phase with every other party, then the encryption scheme is secure, leading to a contradiction.
2. If M' is not related to S, then the adversary has managed to guess a random value from input independent of that value. If there are n_1 elements of $\mathcal{R}$ then the adversary has a $\frac{n_1}{2^n}$ chance of guessing any given nonce. If there are n_2 elements of $\mathcal{K}_{Priv}$ and each key is $l(n)$ bits long, then the adversary has a $\frac{n_2}{2^{l(n)}}$ chance of guessing any private key. Since both of these are negligible, then the adversary must have a negligible change creating an encoding of M'.

Hence, $\frac{p}{q(n)}$ must be negligible, which means that p must have been negligible to begin with. ■

Hence, plaintext-aware encryption limits the computational adversary to the operations available to the Dolev-Yao adversary. It is unknown whether any weaker form of encryption achieves the same limitation, making this the first naturally arising application of plaintext-aware cryptography.

6. Acknowledgments

The authors would like to thank Ron Rivest and Nancy Lynch, under whose supervision part of this work was done. They would also like to thank the anonymous referees for their insightful comments.

References

[1] 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)*, number 1872 in Lecture Notes in Computer Science, pages 3–22. Springer-Verlag, 2000.

[2] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In H. Krawczyk, editor, *Advances in Cryptology (CRYPTO 98)*, volume 1462 of *Lecture Notes in Computer Science*, pages 26–45. Springer–Verlag, 1998. Full version found at http://www.cs.ucsd.edu/users/mihir/papers/relations.html.

[3] Mihir Bellare and Oded Goldreich. On defining proofs of knowledge. In E. Brickell, editor, *Advances in Cryptology – Crypto 92 Proceedings*, volume 740 of *Lecture Notes in Computer Science*, pages 390–420. Springer–Verlang, 1992.

[4] Mihir Bellare and Phillip Rogaway. Optimal asymmetric encryption– how to encrypt with RSA. In A. De Santis, editor, *Advances in Cryptology – Eurocrypt 94 Proceedings*, volume 950 of *Lecture Notes in Computer Science*, pages 92–111. Springer-Verlag, 1995.

[5] M. Blum, P. Feldman, and S. Micali. Non-interactive zero knowledge proof systems and applications. In *Proceedings of the 20th Annual ACM Symposium on Theory of Computing*, pages 103–112, 1988.

[6] M. Blum, A. De Santis, S. Micali, and G. Persiano. Noninteractive zero knowledge. *SIAM Journal on Computing*, 20(6):1084–1118, December 1991.

[7] R Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology — CRYPTO 1998*, number 1462 in LNCS, pages 13–25. Springe–Verlag, 1998.

[8] D. Dolev and A. Yao. On the security of public-key protocols. *IEEE Transactions on Information Theory*, 29:198–208, 1983.
[9] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. *Journal of the ACM*, 38(1):691–729, 1991.
[10] S. Goldwasser and S. Micali. Probabilistic encryption. *Journal of Computer and System Sciences*, (28):270–299, 1984.
[11] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowedge complexity of interactive proof systems. In *Proceedings of the 17th ACM Symposium on Theory of Computing*, pages 291–304, 1985. Superseded by journal version.
[12] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital-signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.
[13] Jonathan Herzog. Computational soundness for formal adversaries. Master's thesis, Massachusetts Institute of Technology, October 2002.
[14] Gavin Lowe. Breaking and fixing the Needham–Schroeder public-key protocol using FDR. In Margaria and Steffen, editors, *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 147–166. Springe–Verlag, 1996.
[15] M. Naor and M. Yung. Public-key cryptosystems provably secure against chosen ciphertext attacks. In *22nd Annual ACM Symposium on Theory of Computing*, pages 427–437, 1990.
[16] L C Paulson. The inductive approach to verifying cryptographic protocols. *Journal of Computer Security*, 6:85–128, 1998.
[17] C. Rackoff and D. Simon. Noninteractive zero-knowledge proof of knowledge and the chosen-ciphertext attack. In *Advances in Cryptology– CRYPTO 91*, number 576 in Lecture Notes in Computer Science, pages 433–444, 1991.
[18] Amit Sahai. Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In *Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 543–553, 1999.
[19] A. De Santis and G. Persiano. Zero-knowledge proofs of knowledge without interaction. In *Proceedings of 33rd IEEE Symposium on Foundations of Computer Science*, pages 427–436, 1992.
[20] Alfredo De Santis, Silvio Micali, and Giuseppe Persiano. Non-interactive zero-knowledge proof systems. In Carl Pomerance, editor, *Proceedings Crypto '87*, pages 52–72. Springer-Verlag, 1988. Lecture Notes in Computer Science No. 293.
[21] D. Song. Athena, an automatic checker for security protocol analysis. In *Proceedings of the 12th IEEE Computer Security Foundations Workshop*, pages 192–202, 1999.

Appendix A. False Simplifications

At first glance, it would seems that both our model and our implementation could be greatly simplified. However, while this may indeed be the case, many innocent-seeming simplification lead to intractable complications later on. For example,

- *A Simplification to the Model* Rather than relying on a registration authority, one may assume the existence of a common, public reference string. Such a string could easily specify a public key PK, and a user U might encrypt a message m both in the recipient's key and in PK, and provide a proof of having done so. Following [19], this process (1) is a non-interactive zero-knowledge proof of knowledge, and (2) can be proven to yield a chosen-ciphertext-secure scheme, but (3) does *not* yield a plaintext-aware scheme![13]
- *An Efficiency Improvement* Our scheme is quite computationally intensive. By contrast, the scheme of [7] is quite efficient and, being based on the Diffie-Hellman system, has naturally embedded the mechanism for decrypting ciphertexts with either the sender or the recipient's secret key. Therefore, to obtain a plaintext-aware scheme with key registration, it seems that all one has to do is (1) making some of the ephemeral keys that a sender generates in the scheme of Cramer and Shoup [7] "permanent" public keys, and (2) have each user provide a zero-knowledge proof of knowledge for a small number of discrete logarithms (which is very efficient to do). Unfortunately, goddess being in the details, at a careful analysis the *provability* of the so modified scheme slips away, and only a much weaker type of plaintext awareness with key registration is readily provable.

[13] As a hint, it may suffice to say that, to really extract the desired knowledge, a simulator would have to replace the reference string with one of his own. But if the ciphertext a malicious sender chooses depended on the reference string, then no such replacement would yield the "original" message.