Plaintext Awareness via Key Registration

Jonathan Herzog, Moses Liskov, and 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: even when the third party is dishonest, we still guarantee security against adaptive chosen ciphertext attacks. 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 [1] (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. Although 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 inventors of plaintext-awareness 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, CCA-2 secure schemes were already known: they were constructed by Cramer and Shoup [3] under the decisional Diffie-Hellman assumption (yielding a very efficient scheme), and by Sahai [4] (improving on previous work of Naor and Yung [5]) under very general complexity assumptions. Thus, genuinely new applications of PA encryption, despite its intuitive great power, have been somewhat scarce.

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 (being already used in practice) 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:* Only the *plaintext awareness* of our scheme depends on the honesty of the registration authority. In particular, the *security* does not. Even if the registration authority collaborated with the adversary, our scheme is guaranteed to be CCA-2 secure.
- *Naturalness:* A trusted registration authority is essentially implicit in any actual implementation of public-key encryption. Such implementations enforce a correct association between users and public keys by requiring that

users register their public key with a *certification authority*. These authorities verify the identity of the applicant and that the applicant knows the corresponding secret key.

In our system, users will have separate keys for sending and receiving messages, and our definitions only require that the sending keys key be registered. However, it is natural to require users to register their sending keys at the same time that they have their receiving keys certified, and that the certificate authority act as registering authority also.

– *Efficiency:* As we've said, a random oracle can be thought of as a trusted third party. However, in the Bellare-Rogaway model, this trusted third party must 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 [6, 4, 5], and makes use of the following key registration process: a sender simply gives a zero-knowledge proof of knowledge of his secret sending key. Since the proof system is zero-knowledge, no registration authority (honest or dishonest) gains any information.

Following [6], we also make the encryption of a message depend on the public keys of both sender and receiver. More precisely, and giving a self-referential twist to the schemes of [5] and [4], our sender U encrypts a message for V both in V's public receiving key as well as his own public sending key — and provides a proof of having done so.

The Essence of Our Application. We apply plaintext awareness to the *Dolev-Yao* model [7], the famous alternative for cryptographic protocol analysis.[1] Unlike the more general computational models, the Dolev-Yao model has the advantage of extreme simplicity and ease of use. Although it is impossible to decide the correctness of a protocol in general, the correctness of an impressive number of specific protocols has been successfully decided by automated tools [8–10].

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.

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

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

We say that an algorithm (or interactive TM) A is *history-preserving* 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 history-preserving algorithm, then if A appears more than once in a piece of GMR notation (e.g, $\Pr[\ldots; a \overset{R}{\leftarrow} A(x); \ldots; b \overset{R}{\leftarrow} A(y); \ldots : p(\cdots, a, b, \cdots)])$ then the history and state of A is preserved from the end of one "use" to the beginning of the next. The notation $h \overset{H}{\longleftarrow} A$ indicates that h is the content of the current history tape of A.

An *adversary* is an efficient history-preserving algorithm (interactive TM).

Following [11], we consider a two-party protocol as a pair, (A, B), of interactive Turing machines. By convention, A takes input (x, r_A) and B takes input (y, r_B) where x and y are arbitrary and r_A and r_B are random tapes. On these inputs, protocol (A, B) computes in a sequence of rounds, alternating between A-rounds and B-rounds. In an A-round only A is active and sends a message (i.e., a string) that will become an available input to B in the next B-round. (Likewise for B-rounds.) A computation of (A, B) ends in a B-round in which B sends the empty message and computes a private output.

If E is an execution of (A, B) on inputs (x, r_A) and (y, r_B), then the *output of A in E* (denoted $OUT_A^{A,B}(x, r_A|y, r_B)$) consists of the string z output by A in the last A-round. Similarly, $OUT_B^{A,B}(x, r_A|y, r_B)$ is the output of B in the same execution. We also define the random distribution $OUT_A^{A,B}(x, \cdot|y, \cdot)$ to be $OUT_B^{A,B}(x, r_A|y, r_B)$ where r_A and r_B are selected randomly.

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 $(1^k, y')$.

3 The Notion of Plaintext Awareness via Key Registration

3.1 Informally

Plaintext-awareness via key registration requires a significantly different definition than those of other plaintext-aware cryptosystems. We insist that not only the receiver of encrypted messages have a public key but also that the sender have a public key, registered in advance with the registration authority. In this setting, plaintext awareness means the following: the adversary can decrypt *any* ciphertext it creates, so long as the (apparent) sender has registered its sending key with the proper registration authority.

Also, we ask that plaintext-awareness hold for any key registered with the honest registration authority. However, as mentioned before, the security of our scheme should not depend on the honesty of the registration authority. The scheme should remain CCA-2 secure (i.e., the most secure possible without a trusted third party) even if the registration authority collaborates with the adversary.

A plaintext-aware encryption scheme consists of an encryption scheme (G, E, D), and a key-registration protocol $(\mathsf{RU}, \mathsf{RA})$.

Algorithm G is used for the generation of the receiver's encryption and decryption keys. In this model, E and D must also be given the public key of the sender as input.

The sender must participate in the key-registration protocol in order to generate a key. The registration is performed by having U run protocol RU on input 1^k with the registration authority running RA on input 1^k. If the registration is successful, the registration authority simply outputs the key e_s, and U should also output this key. One can think of the key as then being inserted in a public file or that U is given a certificate for e_s, but the precise mechanism of the publication 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.[2]

It is worth noting that either RU or RA may reject in the registration protocol (presumably when the other party is dishonest), in which case we assume the output is $\perp$. For ease in the definitions, we assume that if $\perp$ is any input to either E or D, the output will also be $\perp$.

3.2 More Formally

A *registration-based plaintext-aware encryption scheme* consists of a pair (G, E, D) and (RU, RA), where

- (G, E, D) is a public-key encryption scheme, where:
  - $\mathsf{G}(1^k)$ produces (e_r, d_r), a key pair for the receiver, where k is a security parameter;
  - $\mathsf{E}(m, e_r, e_s)$ produces c, where c is a ciphertext, m is the message to encrypt, and e_r and e_s are the receiver's and sender's public keys; (The ciphertext c is assumed to explicitly indicate which public keys were used in its creation.)

[2] 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.

- $\mathsf{D}(c, d_r, e_s)$ produces m, a message, where c is the ciphertext to decrypt, d_r is the receiver's private key, and e_s is the sender's public key. (If the ciphertext is invalid, the output is $\perp$.)

– $(\mathsf{RU}, \mathsf{RA})$ is a two-party protocol in which both parties should output e_s, a public key for the sender;

which satisfy the following conditions (in which ν is a negligible function):

– *Registration Completeness*: The key registration protocol between an honest registrant and an honest registration authority will almost always be successful, and the user and the authority will agree on the key.

$$\begin{array}{ll} \forall k & \\ \Pr[\; r_1 \xleftarrow{R} \{0,1\}^*; r_2 \xleftarrow{R} \{0,1\}^*; & \\ \quad e_s \xleftarrow{R} OUT_{\mathsf{RA}}^{\mathsf{RU},\mathsf{RA}}\left(1^k, r_1 | 1^k, r_2\right); & \\ \quad e_s' \xleftarrow{R} OUT_{\mathsf{RU}}^{\mathsf{RU},\mathsf{RA}}\left(1^k, r_1 | 1^k, r_2\right); & \\ \quad e_s = e_s' \neq \perp &] = 1 - \nu(k) \end{array}$$

– *Encryption Completeness*: If an honest sender encrypts a message m into a ciphertext c, then the honest recipient will almost always decrypt c into m.

$$\begin{array}{ll} \forall k, \forall m \in \{0,1\}^k & \\ \Pr[\; e_s \xleftarrow{R} OUT_{\mathsf{RU}}^{\mathsf{RU},\mathsf{RA}}\left(1^k, \cdot | 1^k, \cdot\right); & \\ \quad (e_r, d_r) \xleftarrow{R} \mathsf{G}(1^k); & \\ \quad c \xleftarrow{R} \mathsf{E}(m, e_r, e_s); & \\ \quad g \xleftarrow{R} \mathsf{D}(c, d_r, e_s): & \\ \quad g = m &] = 1 - \nu(k) \end{array}$$

– *Honest Security:* If recipient and sender are honest, the encryption is adaptively chosen-ciphertext secure even if the adversary controls the registration authority.

$$\begin{array}{ll} \forall \text{ oracle-calling adversaries } \mathsf{A}, \forall \text{ sufficiently large } k & \\ \Pr[\; (d_r, e_r) \xleftarrow{R} \mathsf{G}(1^k); & \\ \quad e_s \xleftarrow{R} OUT_{\mathsf{RU}}^{\mathsf{RU},\mathsf{A}}\left(1^k, \cdot | 1^k, \cdot\right); & \\ \quad m_0, m_1 \xleftarrow{R} \mathsf{A}^{D(\cdot, d_r, \cdot)}(e_r, e_s); & \\ \quad b \xleftarrow{R} \{0,1\}; & \\ \quad c \xleftarrow{R} \mathsf{E}(m_b, e_r, e_s); & \\ \quad g \xleftarrow{R} \mathsf{A}^{D(\cdot, d_r, \cdot) - \{c\}}(c): & \\ \quad b = g &] \leq \frac{1}{2} + \nu(k) \end{array}$$

where

- m_0 and m_1 have the same length, and
- $D(\cdot, d_r, e_s) - \{c\}$ is the oracle that returns $\mathsf{D}(c', d_r, e_s)$ if $c' \neq c$ and returns $\perp$ if $c' = c$.

Note that if $e_s = \perp$ the adversary will get only $\perp$ from the oracle, and $c = \perp$ as well, so the probability of success will be just 1/2. Also, recall that the adversary is assumed to be history-preserving, so that it remembers every input it has ever seen.

– *Plaintext Awareness:* If the registration authority is honest and player X (either the adversary or an honest player) registers a key, then the adversary can decrypt any string it sends to an honest participant ostensibly by X:

$$\begin{array}{l} \forall \text{ adversaries } \mathsf{A}, \forall \mathsf{X} \in \{\mathsf{A}, \mathsf{RU}\}, \exists \text{ efficient algorithm } \mathsf{S}_\mathsf{X}, \forall \text{ s.l. } k \\ \Pr[\ (e_r, d_r) \stackrel{R}{\leftarrow} \mathsf{G}(1^k); \\ \quad e_\mathsf{X} \stackrel{R}{\leftarrow} OUT_{\mathsf{RA}}^{\mathsf{X},\mathsf{RA}}\left(e_r, \cdot | 1^k, \cdot\right); \\ \quad h \stackrel{H}{\longleftarrow} \mathsf{A}; \\ \quad c \stackrel{R}{\leftarrow} \mathsf{A}^{\mathsf{D}(\cdot, d_r, \cdot)}(e_\mathsf{X}, e_r); \\ \quad \mathsf{S}_\mathsf{X}(h, c, e_r, e_\mathsf{X}) = \mathsf{D}(c, d_r, e_\mathsf{X})\ ] \geq 1 - \nu(k) \end{array}$$

Remarks. Note in the definition of plaintext awareness that if $\mathsf{X} = \mathsf{RU}$, then it expects its input to be 1^k and not e_r. Hence, we assume that if RU finds input e_r that it extracts 1^k from it and proceeds as normal.

Also in the definition of plaintext-awareness, if the sender key is registered by an honest participant ($\mathsf{X} = \mathsf{RU}$) then h, the history of the adversary, will be empty.

Lastly, note that these definitions do not guarantee anonymity of the sender. That is, senders must register their keys, and so it might be that they can no longer send messages without their name attached in some way. We note three things with respect to this.

1. If plaintext-awareness is not required, a sender may simply use an unregistered key. Plaintext awareness will no longer be guaranteed, but chosen ciphertext security will still hold.
2. Each registered key does not necessarily represent a sender but rather one incarnation of a sender. Senders may register many keys in order to bolster their anonymity.
3. Lastly, we note that in our motivating application, channels will be authenticated anyway, so this is no additional loss. Indeed, authentication is almost necessary, as our definition guarantees only that a message encrypted under a registered key will be known to the party that registered that key.

We choose to regard the possibility of sender authentication as an opportunity rather than a drawback, and use it in an essential way in our implementation.

4 Implementing Plaintext Awareness via Key Registration

Our scheme uses non-interactive zero-knowledge proofs (e.g. [12–15]) in order to enhance encryption security. This approach has been pioneered by Naor and

Yung [5], and greatly refined by Sahai [4]. Another fount of inspiration comes from the work of Rackoff and Simon [6] which used a very powerful registration authority (indeed, one that chooses every user's secret keys) to obtain chosen-ciphertext security.

We make use of the following three cryptographic tools:

- $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$, a semantically secure cryptosystem in the sense of [16].
- (f, P, V, S), a non-malleable NIZK proof system for NP in the sense of [4], 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, and [17, 18, 11].
- Authenticated channels that allow a recipient to determine if a ciphertext (c, e, e') was sent by the entity that registered the sending key e'.

The first three of the above rely only upon the existence of trapdoor permutations. The authenticated channels may introduce additional assumptions.

4.1 The Scheme $\mathcal{S}$

The scheme $\mathcal{S} = (\mathcal{G}, \mathcal{E}, \mathcal{D}, \mathcal{RU}, \mathcal{RA})$ is as follows.[3]

- $\mathcal{G}$ (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 $e_r = (e_1, e_2, \sigma)$ and the secret key is $d_r = (d_1, d_2)$.
- $\mathcal{RU}$ and $\mathcal{RA}$: First, Generate (e_3, d_3) according to $\mathsf{G}'(1^k)$. The public (sender's) key is $e_s = e_3$. Next, we 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 $\perp$. $\mathcal{RU}$ outputs e_s so long as the zero-knowledge proof of knowledge was not aborted, otherwise it outputs $\perp$.
- $\mathcal{E}$, 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).
- $\mathcal{D}$, on input $((c_1, c_2, c_3, \pi), (e_1, e_2, \sigma, d_1, d_2), (e_3))$ first determines if the ciphertext (c_1, c_2, c_3, π) was sent by the entity that registered e_3. (Authenticated channels are essential for this step.) If not, it outputs $\perp$. If so, it then determines if π is a valid proof that c_1, c_2 and c_3 are encryptions of the same message 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 $\perp$.

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

4.2 Security of $\mathcal{S}$

$\mathcal{S}$ Satisfies Registration Completeness. This is natural: the registration process is a zero-knowledge protocol. By its completeness property, an honest prover will almost always be able to prove a true theorem (e_s) to an honest verifier if it possesses a witness (d_s). Since the honest registrant has access to the witness and engages in the protocol honestly, the honest registration authority will almost always accept the proof and output the public key, and the user will output the same key.

$\mathcal{S}$ Satisfies Encryption 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 almost always accept it and decrypt c_1 to receive m, the same message encrypted by the sender.

$\mathcal{S}$ Satisfies 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.

R simulates the adversary A so as to break the semantic security of the underlying encryption scheme $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$. So, 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[4]. 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'), encrypted with sending key e', 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 $c_a \stackrel{R}{\leftarrow} \mathsf{E}'(e_a, m_b)$, and set $c_{3-a} \stackrel{R}{\leftarrow} c$. With probability $1/2$, let $c_3 \stackrel{R}{\leftarrow} \mathsf{E}'(e_3, m_b)$ and otherwise, let $c_3 \stackrel{R}{\leftarrow} \mathsf{E}'(e_3, m_{1-b})$. Fake the NIZK proof π using the simulator S.
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.

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

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 x be such 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. 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, the oracle will return $\perp$ if the adversary ever makes a decryption query when c_1 and c_2 encrypt different messages. Thus, the case $a = 1$ and the case $a = 2$ give the same distribution. (See [4]. This is just like one of the main details from Sahai's proof that his scheme is CCA2-secure.)

   Let $m_{\beta'}$ be the message encrypted in c_3, and let y be such that in this case the adversary returns β'.

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

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

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 $(\mathsf{G}', \mathsf{E}', \mathsf{D}')$. 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 that works as follows, on input e:

1. Generate (e_1, d_1) and (e_2, d_2) by running $\mathsf{G}'(1^k)$. Generate σ according to the simulator S for the NIZK proof system. Set e_3 to 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 (which we assume to be history-preserving).
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 $1/2 + \epsilon - |\nu'/2|$.

***$\mathcal{S}$* Enjoys Plaintext Awareness.** This is fairly simple, and we show it informally. There are two cases. If $X = \mathsf{RU}$, then e_{X} was registered by an honest user. Hence, when the adversary creates a ciphertext ostensibly from that honest user, it will fail to decrypt. (The use of authenticated channels will tell the receiver that it was sent by the adversary, and not the entity that registered e_{X}.) Hence, S_{X} simply outputs $\perp$ on all input.

In the other case, $X = \mathsf{A}$, and the adversary registered e_{X}. We extract plaintext as follows. On input $(h, (c_1, c_2, c_3, \pi), e_r, e_{\mathsf{X}})$, we use h to rewind the adversary to the point where A engages in key registration with RA. We then use the extractor from the interactive zero knowledge proof of knowledge to find a value d, the secret key associated with e_{X}. We then check π; if π is invalid, we output

$\perp$. Otherwise, we use d to decrypt c_3 and give the result as the answer. From the extractibility property of the proof system, d must be a secret key relative to *key* so this answer is correct.

However, we do need to show that the decryption under d will always be the same as the decryption under d_r. If the proof π in c is invalid, then certainly we are correct to output $\perp$. If π is valid, then by the soundness of the NIZK proof system, it must be that c_1, c_2, and c_3 all encrypt the same message, so we are still 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 [7].

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.

In this model, messages are not bit-strings but parse trees. The atomic elements (leaves) are considered to be abstract symbols with no internal structure, and are partitioned into three sets: names ($\mathcal{M}$), random numbers ($\mathcal{R}$) or keys ($\mathcal{K}_{Pub}$ and $\mathcal{K}_{Priv}$).[5] Compound messages are formed using two operations:

– $\mathit{encrypt} : \mathcal{K}_{Pub} \times \mathcal{A} \to \mathcal{A}$
– $\mathit{pair} : \mathcal{A} \times \mathcal{A} \to \mathcal{A}$

We denote $\mathit{encrypt}(K, M)$ by $\{\!|M|\!\}_K$, and denote $\mathit{pair}(M, N)$ by $M\,N$. We denote by $\mathcal{A}$ the set of all messages. Because messages are parse trees, every message has a unique interpretation. We assume for our purposes that the algebra 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 adversary 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:

[5] We will only consider the case of asymmetric encryption, though the Dolev-Yao models symmetric encryption also.

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 [19] 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 $(\mathsf{G}, \mathsf{E}, \mathsf{D})$.[6] 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
- 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 $\mathsf{E}([\![M_1]\!]_n, [\![K]\!]_n)$.

[6] Our definition of plaintext-aware encryption contains several more algorithms, which we ignore for the moment.

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

$$\begin{array}{l} \forall\, \mathsf{A}_{PPT},\ \forall\, S \subseteq \mathcal{A},\ \forall\, M \notin C\,[S]\,,\ \forall \textit{ polynomials } q,\ \forall \textit{ sufficiently large } n: \\ \Pr[\ s \xleftarrow{R} [S \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Subv} \cup \mathcal{M}]_n\,; \\ \quad m \xleftarrow{R} \mathsf{A}(1^n, s): \\ \quad m \in \boldsymbol{supp}\,[M]_n \qquad\qquad] \leq \frac{1}{q(n)} \end{array}$$

(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 [20]. 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.[7].

Hence, we revise the security condition:

Definition 4 *An encryption scheme* $(\mathsf{G}, \mathsf{E}, \mathsf{D})$ *is* ideal *if the adversary cannot create something outside the closure:*

$$\begin{array}{l} \forall\, \mathsf{A}_{PPT},\ \forall \textit{ acyclic } S \subseteq \mathcal{A},\ \forall\, M \notin C\,[S]\,,\ \forall \textit{ polynomials } q,\ \forall \textit{ sufficiently large } n: \\ \Pr[\ s \xleftarrow{R} [S \cup \mathcal{K}_{Pub} \cup \mathcal{K}_{Subv} \cup \mathcal{M}]_n\,; \\ \quad m \xleftarrow{R} \mathsf{A}(1^n, s): \\ \quad m \in \boldsymbol{supp}\,[M]_n \qquad\qquad] \leq \frac{1}{q(n)} \end{array}$$

This definition, it turns out, is no stronger than plaintext-awareness. Before we can prove this, however, we need to address a small technical issue. Encryption in the Dolev-Yao model depends only on the message and the receiver's public key. In our definition of plaintext-aware encryption, however, encryption uses the receiver's public receiving key and the sender's public sending key. Hence, we consider a slight variant of the Dolev-Yao model in which the formal encryption operation uses two public keys: the sender's and the receiver's. The encoding of a formal key contains both a sending portion and a receiving portion (public or private, as appropriate). The encoding of a formal encryption is then defined using, in the natural way, a computational plaintext-aware encryption scheme and the encodings of the public keys.

Theorem 5. *Any encryption scheme that achieves 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 non-negligible probability the adversary could create an encoding m such that m is the valid encoding of an M not in $C\,[S]$.

[7] This is a reasonable assumption for most "real-world" protocols, for reasons discussed in [20]

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

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, then plaintext-awareness tells us that there exists a simulator that can extract the plaintext (which by construction, is never $\perp$.) Hence, the adversary can create the encoding of an encryption, then run the simulator to extract the plaintext of the encryption. Also, since all public keys are known to the adversary, it can create both encryption keys. Thus, the adversary can create all children of a encryption node.

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. Bellare, M., Rogaway, P.: Optimal asymmetric encryption– how to encrypt with RSA. In Santis, A.D., ed.: Advances in Cryptology – Eurocrypt 94 Proceedings. Volume 950 of Lecture Notes in Computer Science., Springer-Verlag (1995) 92–111
2. Bellare, M., Desai, A., Pointcheval, D., Rogaway, P.: Relations among notions of security for public-key encryption schemes. In Krawczyk, H., ed.: Advances in Cryptology (CRYPTO 98). Volume 1462 of Lecture Notes in Computer Science., Springer–Verlag (1998) 26–45 Full version found at http://www.cs.ucsd.edu/users/mihir/papers/relations.html.
3. Cramer, R., Shoup, V.: A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In: Advances in Cryptology — CRYPTO 1998. Number 1462 in LNCS, Springer–Verlag (1998) 13–25
4. Sahai, A.: Non-malleable non-interactive zero knowledge and adaptive chosen-ciphertext security. In: Proceedings of 40th Annual IEEE Symposium on Foundations of Computer Science (FOCS). (1999) 543–553
5. Naor, M., Yung, M.: Public-key cryptosystems provably secure against chosen ciphertext attacks. In: 22nd Annual ACM Symposium on Theory of Computing. (1990) 427–437
6. Rackoff, C., Simon, D.: 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 (1991) 433–444
7. Dolev, D., Yao, A.: On the security of public-key protocols. IEEE Transactions on Information Theory **29** (1983) 198–208
8. Lowe, G.: Breaking and fixing the Needham–Schroeder public-key protocol using FDR. In Margaria, Steffen, eds.: Tools and Algorithms for the Construction and Analysis of Systems. Volume 1055 of Lecture Notes in Computer Science. Springer–Verlag (1996) 147–166
9. Paulson, L.C.: The inductive approach to verifying cryptographic protocols. Journal of Computer Security **6** (1998) 85–128
10. Song, D.: Athena, an automatic checker for security protocol analysis. In: Proceedings of the 12th IEEE Computer Security Foundations Workshop. (1999) 192–202
11. Goldwasser, S., Micali, S., Rackoff, C.: The knowedge complexity of interactive proof systems. In: Proceedings of the 17th ACM Symposium on Theory of Computing. (1985) 291–304 Superseded by journal version.
12. Blum, M., Feldman, P., Micali, S.: Non-interactive zero knowledge proof systems and applications. In: Proceedings of the 20th Annual ACM Symposium on Theory of Computing. (1988) 103–112
13. Santis, A.D., Micali, S., Persiano, G.: Non-interactive zero-knowledge proof systems. In Pomerance, C., ed.: Proceedings Crypto '87, Springer-Verlag (1988) 52–72 Lecture Notes in Computer Science No. 293.
14. Blum, M., Santis, A.D., Micali, S., Persiano, G.: Noninteractive zero knowledge. SIAM Journal on Computing **20** (1991) 1084–1118

15. Boyar, J., Damgård, I., Peralta, R.: Short non-interactive cryptographic proofs. Journal of Cryptology: the journal of the International Association for Cryptologic Research **13** (2000) 449–472
16. Goldwasser, S., Micali, S.: Probabilistic encryption. Journal of Computer and System Sciences (1984) 270–299
17. Bellare, M., Goldreich, O.: On defining proofs of knowledge. In Brickell, E., ed.: Advances in Cryptology – Crypto 92 Proceedings. Volume 740 of Lecture Notes in Computer Science., Springer–Verlang (1992) 390–420
18. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity or all languages in NP have zero-knowledge proof systems. Journal of the ACM **38** (1991) 691–729
19. Abadi, M., Rogaway, P.: 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, Springer-Verlag (2000) 3–22
20. Herzog, J.: Computational soundness for formal adversaries. Master's thesis, Massachusetts Institute of Technology (2002)