Modelling Cryptographic Keys in Dynamic Epistemic Logic with DEMO

Hans van Ditmarsch, Jan van Eijck, Ignacio Hernández-Antón, Floor Sietsma, Sunil Simon, and Fernando Soler-Toscano

Abstract It is far from obvious to find logical counterparts to crytographic protocol primitives. In logic, a common assumption is that agents are perfectly rational and have no computational limitations. This creates a dilemma. If one *merely* abstracts from computational aspects, protocols become trivial and the difference between tractable and intractable computation, surely an essential feature of protocols, disappears. This way, the protocol gets lost. On the other hand, if one '*merely*' (scare quotes indeed) models agents with computational limitations (or otherwise bounded rationality), very obvious aspects of reasoning become problematic. That way, the logic gets lost. We present a novel way out of this dilemma. We propose an abstract logical architecture wherein public and private, or symmetric keys, and their roles in crytographic protocols, all have formal counterparts. Instead of having encryption and decryption done by a principal, the agent sending or receiving messages, we introduce additional, virtual, agents to model that, so that one-way-function aspects of computation can be modelled as constraints on the communication between principals and these virtual counterparts. In this modelling it does not affect essential protocol features if agents are computationally unlimited. We have implemented the proposal in a dynamic epistemic model checker called DEMO.

1 Introduction

The security of many protocols depends on the computational limitations of the agents involved and on the intractability of inverses for quite tractable computations. *A* standard example is public/private key encryption, e.g., the security of the RSA protocol [6] is (also) based on the complexity of integer factorization. It is easy to multiply primes. It is hard to factorize a number into its prime constituents.

H. van Ditmarsch, I. Hernández, F. Soler, University of Sevilla, Spain. Email: {hvd,iha, fsoler}@us.es. J. van Eijck, F. Sietsma, S. Simon, CWI, Amsterdam, Netherlands. Email: {Jan.van.Eijck,S.E.Simon,F.Sietsma}@cwi.nl.

A goal in information theoretic security is to find abstract notions for cryptographic primitives such as keys and one-way-functions, and there have been several proposals for logical modelling of such primitives [5, 4, 1]. In such approaches it is problematic that logical agents know all logical consequences of their information. If they know two primes, they know their product; and if they know a number, they know if it is the product of primes and what these primes are. There are no one-way-functions in logic.

We present a novel proposal to tackle this problem in the setting of dynamic epistemic logic [2], where the knowledge of the agents involved in protocol execution, including higher-order features (what agents know about each other), is represented in relational structures called multi-agent Kripke models, and change of knowledge is represented by various structural transformations. In this setting agents are also computationally unlimited. However, by means of introducing virtual coding and virtual decoding agents we can simulate computational bounds as communicative restrictions between the authentic agents participating in protocols (the principals) and these virtual agents.

Given a sender *A* (Alice) and receiver *B* (Bob), we introduce a coding agent *C* (Coder) and a decoding agent *D* (Decoder). The eavesdropper *E* (Eve) listens in to public communications. For the specific setting of asymmetric public/private key encryption, every agent can communicate in a certain way with the coding agent, who represents the public key, but only one, *B*, will be able to communicate with the decoding agent, who represents the private key. We can think of the coding and decoding agents as follows. We split each agent into two parts, its *knowledge base*, where we will evaluate the agent's knowledge, and its *computational resources* (cryptographic functions). The first is considered the principal in protocol execution and the latter the virtual agent counterpart. Both can be assumed to be computationally unlimited (this is not a requirement but the assumption to the contrary is what makes *logical* modelling of security so hard and counterintuitive). The coding is performed by *C* and not by *A*; the decoding is performed by *D* and not by *B* (and both in a way to which no other agent is privy). What the non-(de)coding agents get to know about the outcome of that process is determined by communication, not by computation. On the other hand, the coding and decoding agent only perform that virtual role. They are not privy to the standard communicative aspects of the protocol.

Next, we present the protocol, illustrated by a two-bit secure message passing. Then, an adaptation to symmetric key encryption, and an application, RSA.

2 Public/private-key encryption with coding and decoding agents

- *A*: Alice (sender)
- *B*: Bob (receiver)
- *C*: Coding agent (Bob's public key)
- *D*: Decoding agent (Bob's private key)
- *E*: Eve (eavesdropper)

We distinguish five agents. The coding and decoding agents are the *cryptographic agents* and can occur in various protocols. In public-private key encryption the coding agent C stands for Bob's public key and the decoding agent D represents Bob's private key. The agents A, B and E are the *principals* of the protocol (where E may not always be that passive). Principals are proactive agents while cryptographic agents C and D are reactive. We model the knowledge and ignorance of all these agents, represented as uncertainty between the valuations of variables (bits).

The protocol consists of an initialization phase and an execution phase. Information is a bitstring and uncertainty about information is represented as alternative bitstrings. A message can be an *individual message* (to one agent, a.k.a. private – but that would be confusing in our setting with private keys), a *group message* (to more than one and less than five agents), and a *public message* (to all five agents, a.k.a. a public announcement). In the initialization phase background knowledge is incorporated, e.g., that the encoding agent C knows the encryption function. The execution phase consists of the encryption and decryption; its interest to the reader consists of its communicative aspects and properties. As a running example Alice will communicate two bits to Bob.

Clear		**Encoded**	
Number	**Formula**	**Number**	**Formula**
0	$\neg p_1 \wedge \neg p_0$	1	$\neg q_1 \wedge q_0$
1	$\neg p_1 \wedge p_0$	3	$q_1 \wedge q_0$
2	$p_1 \wedge \neg p_0$	0	$\neg q_1 \wedge \neg q_0$
3	$p_1 \wedge p_0$	2	$q_1 \wedge \neg q_0$

Table 1 Correspondence between clear and encoded messages

With two bits there are four possible secrets. They are shown in Table 1, and also the encoded message that corresponds to each secret. We use p_i to represent *clear messages* and q_i for *encoded messages*. We also use r_i atoms for clear messages. So for example message number 2 is not just $p_1 \wedge \neg p_0$ but $p_1 \wedge \neg p_0 \wedge r_1 \wedge \neg r_0$. We duplicate each p_i with r_i. This is a technical trick to make that the cryptographic agents C and D have different information from the other agents.

In the initialization phase of the protocol (a preprocessing of the relational model to represent the knowledge at the start of protocol execution), A is given a secret value of p_0 and p_1, say 2; and principals A, B and E will be made aware that, for each $i \in \{0,1\}$, p_i and r_i are equivalent (or, in general, that each p_i is either equivalent to r_i or to $\neg r_i$), but not C and D.

What C and D know is shown in Figure 1. Agent C knows the encoding that corresponds to each clear message. Agent D has the information to decode every encoded message. The trick of using both p_i and r_i is that the information of C and D is different, as required for a one-way function. Principals A, B and E will receive a *group message* that $p_i \leftrightarrow r_i$ for $i \in \{0,1\}$. But C and D don't know the correspondence between these two sets of atoms. We can now make a difference between public and private keys.

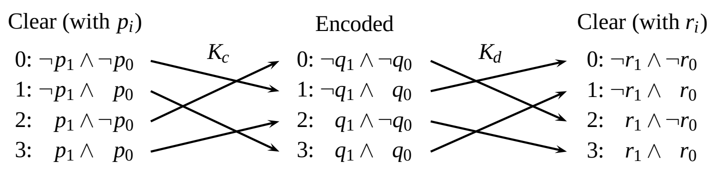

Fig. 1 Encryption and decryption of two bits of information

We now get to the execution phase of the protocol. Agent C is Bob's public key. Any of the principals A, B or E can send it a private message consisting of values for p_0 and p_1. Consequently, encoding agent C, who is the only agent knowing the correspondence of these values to values for q_0 and q_1, makes a public announcement of these values (a *public message*). Agent D is Bob's private key. It reacts when a public announcement has been made about q_0 and q_1 by way of informing *only B* (and not just anyone — this is the *private* key part) with a private message about r_0 and r_1. The workflow for A to communicate B the secret 2 is shown in Figure 2.

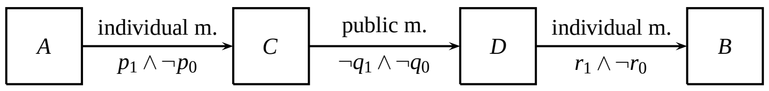

Fig. 2 Alice communicates Bob the number 2

3 DEMO implementation

The protocol is implemented in the model checker DEMO [3]. We have employed DEMO Light[1]. First, we declare a module with imported libraries (omitted). As we are going to encrypt two bits, there are four possible secrets. We define a general function `atom` that returns the representation of a given number (in `[0..3]`) with a given propositional letter (`P`, or `Q`). It is the same representation as in Table 1. We also define the logical operator of equivalence `equiv`.

```
atom :: (Num t, Num t1) => (t1 -> Prp) -> t -> Form
atom p 0 = Conj[(Neg (Prp (p 1))), (Neg (Prp (p 0)))]
atom p 1 = Conj[(Neg (Prp (p 1))), (Prp (p 0))]
atom p 2 = Conj[(Prp (p 1)), (Neg (Prp (p 0)))]
atom p 3 = Conj[(Prp (p 1)), (Prp (p 0))]
equiv :: Form -> Form -> Form
equiv a b = Conj[(impl a b), (impl b a)]
```

- **Step 0: Initialization**

[1] See `http://homepages.cwi.nl/˜jve/software/demolight0/`

We explain the code: `relation_p_r` contains the relation between the Ps and Rs that is given to the principals *A*, *B* and *E*. The list `secret_pairs` contains all pairs with a secret and its encoding, as in Table 1. The `coding_formula` contains the information that is given to the coding agent, and `decoding_formula` to the decoding one, as in Figure 1.

```
relation_p_r :: Form
relation_p_r =
  Conj[(equiv (atom P i) (atom R i)) | i<-[0..3] ]
secret_pairs :: [(Integer, Integer)]
secret_pairs  = [(0,1),(1,3),(2,0),(3,2)]
coding_formula :: Form
coding_formula =
  Conj[(impl (atom P (fst i)) (atom Q (snd i))) |
       i<-secret_pairs ]
decoding_formula :: Form
decoding_formula =
  Conj[(impl (atom Q (snd i)) (atom R (fst i))) |
       i<-secret_pairs ]
```

The model of so-called blissful ignorance (common knowledge of ignorance of the values of all variables) is updated by giving each agent the proper information.

```
e1 :: EpistM Integer
e1 = initM [a,b,c,d,e] [P 0, P 1, Q 0, Q 1, R 0, R 1]
iniM :: EpistM Integer
iniM = upds e1
 [ -- Principals are informed about pi<->ri
   groupM [a,b,e] relation_p_r,
   -- agent C is informed about the coding function
   message c coding_formula,
   -- agent D is informed about the decoding function
   message d decoding_formula]
```

Next, executing the protocol. Alice sends Bob the message number 2, by encrypting it with Bob's public key. As Figure 1 shows, the encoding of 2 is 0. The following formulas represent the chosen secret `sec`, the versions with only Ps and Rs and the encrypted message `secE`.

```
sec  = Conj[secP, secR]  -- secret to be communicated
secP = atom P 2          -- secret with Ps
secR = atom R 2          -- secret with Rs
secE = atom Q 0          -- encoded secret
```

To demonstrate the execution of the protocol, we show the updates in the epistemic model and the relevant formulas to be checked. Note that we do not announce formulas with the knowlege operators K. Announcing (in a public or individual message) $K_a\alpha$ gives more information than announcing just α, it also authenticates the information α as coming from agent a. Although authentication is crucial in cryptography, here we just send non-authenticated messages and leave authentication for future work.

- **Step 1: Alice chooses the secret message**

We create the `s1` model by sending Alice the secret information. This way we represent the action of choosing number 2. We check that after this action Alice knows the secret information.

```
s1 :: EpistM Integer
s1 = upd iniM (message a sec)
c1 = isTrue s1 (K a sec)
```

- **Step 2: Alice calls the coding function**

Alice sends the secret message to the coding function in order to encrypt it. We check that the coding agent knows the encoding message.

```
s2 :: EpistM Integer
s2 = upd s1 (message c secP)
c2 = isTrue s2 (K c secE)
```

- **Step 3: The encoding of the message is publicly announced**

The coding agent announces the encrypted message. We check that the decoding agent knows the `R`-part of the secret.

```
s3 :: EpistM Integer
s3 = upd s2 (public secE)
c3 = isTrue s3 (K d secR)
```

- **Step 4: Bob calls the decoding function to learn the secret**

We represent this step with a private message to Bob with the `R`-part of the secret. We have three conditions to check:

1. Bob knows the secret.
2. The ignorance of eavesdropper E is common knowledge to the principals.
3. The secret is not common knowledge to A and B.

```
s4 :: EpistM Integer
s4 = upd s3 (message b secR)
c4 = maybe_And[c41,c42,c43]
c41 = isTrue s4 (K b sec)
c42 = isTrue s4 (CK [a,b,e] (Neg (K e sec)))
c43 = isTrue s4 (Neg (CK [a,b] sec))
```

Given uncertainty about communication channels, it is usual in cryptographic protocols that common knowledge is never obtained. For example, agent A doesn't know that B knows the secret, as A is not sure that B has the key and has used it to decode the message.

4 Simplifications and an Application: RSA encryption

We have modelled public/private key encryption with a coding and decoding agent. That way we simulated one-way functions. In that sort of encryption there is an owner of the key pair, *B*, who has access to both keys. That means that we can do away with agent *D* and give *B* the information of the key, modelled as agent *C*. This then also removes the need to duplicate `P` atoms as `R` atoms. We can then just use `P` atoms for secrets and `Q` atoms for encoded messages. If we remove both cryptographic agents and give the same key to *A* and *B*, we can model symmetric encryption.

As an application we now sketch how to model (for a simple numerical example) the implementation of RSA encryption in dynamic epistemic logic, with DEMO, using the simplified approach with only a coding agent *C*, as above.

Choose primes 3 and 11, so the modulo of the keys is $n = 3 \cdot 11 = 33$ and $\varphi(n) = (3-1)(11-1) = 20$. Possible key pairs are given by pairs (i, j) of integers in $[2, \varphi(n) - 1]$ such that $ij \mathsf{mod} \varphi(n) = 1$. We choose the pair $(13, 17)$. Then the public key (i,n) is $(13,33)$, and the private (j,n) is $(17,33)$. With a public key (i,n), the encoding of a message m, for $0 \leq m < n$, is given by $m^i \mathsf{mod} n$. It is decoded by $(m^i)^j \mathsf{mod} n = m$. For example, with our public key $(13,33)$ we encode the message 15 as $15^{13} \mathsf{mod} 33 = 9$. It can be decoded with the private key $(17,33)$: $9^{17} \mathsf{mod} 33 = 15$.

We now show part of the DEMO code. The list `secret_pairs` contains all possible secret messages and their encoded versions.

```
secret_pairs :: [(Int, Int)]
secret_pairs =  [(i, iˆ13 ‘mod‘ 33) | i<-[0..32]]
```

Clear messages are represented by p_m atoms and encoded messages by q_m, for $0 \leq m < n$. We are not using the binary representation as in the previous section.

The ignorance model `e1` contains all possible combinations of the *P*s and *Q*s, now just one `P` and one `Q` in each state. It is updated with the same action as in the previous section, to obtain `iniM`.

```
e1 :: EpistM Integer
e1 =
  let stats =  -- Possible pairs of Ps and Qs
   zip [0..] [ (i, j) | i<-[0..32], j<-[0..32]] in Mo
  [0..((33ˆ2)-1)]  -- States
  [a,b,c,e]        -- Agents
  ([P i | i<-[0..32]]++[Q i | i<-[0..32]]) -- Props.
  [(w,[P (fromIntegral a0), Q (fromIntegral a1)]) |
   (w, (a0, a1)) <- stats] -- Content of each state
  [(x,i,j) | x<-[a,b,c,e], -- Accessibility relations
   i<-[0..((33ˆ2)-1)], j<-[0..((33ˆ2)-1)]]
  [0..((33ˆ2)-1)]  -- All states are initially pointed
iniM :: EpistM Integer
iniM = upds e1
 [message c coding_formula,message b decoding_formula]
```

We then execute and check the protocol. Alice sends Bob as message the number 15, encrypting it with Bob's public key. The encryption of 15 is 9. The chosen secret `sec` and the encrypted version `secE` are therefore:

```
sec  = Prop (P 15) -- secret (with P)
secE = Prop (Q  9) -- encoded secret (with Q)
```

The same security conditions as in the previous section are then checked.

5 Conclusions

As counterparts of a sender and a receiver we proposed a virtual coding and decoding agent, to simulate encryption and decryption. Computational restrictions (one-way-functions) can be modelled as constraints on the communication between the principals and these virtual counterparts. This agent-based architecture allows us to verify not only knowledge properties of the principals, but also their ignorance. These are surprising but highly desirable results in a logical setting, where all agents are computationally unlimited.

When using our approach to check protocols with large keys we will have to face the problem of representing epistemic models without the need of creating one state for each possible pair of coding/decoding function, as our DEMO implementation (which uses Kripke models) requires. It will increase the efficiency of our approach.[2]

References

1. Dechesne, F., Wang, Y.: To know or not to know: epistemic approaches to security protocol verification. Synthese **177**, 51–76 (2010)
2. van Ditmarsch, H., van der Hoek, W., Kooi, B.: Dynamic Epistemic Logic, *Synthese Library*, vol. 337. Springer (2007)
3. van Eijck, J.: DEMO — a demo of epistemic modelling. In: J. van Benthem, D. Gabbay, B. Löwe (eds.) Interactive Logic — Proceedings of the 7th Augustus de Morgan Workshop, pp. 305–363. Amsterdam University Press (2007). Texts in Logic and Games 1
4. Pucella, R., Halpern, J.: Modeling adversaries in a logic for security protocol analysis (2002). In Formal Aspects of Security, 2002 (FASec '02).
5. Ramanujam, R., Suresh, S.P.: Information based reasoning about security protocols. Electr. Notes Theor. Comput. Sci. **55(1)** (2001)
6. Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM **21(2)**, 120–126 (1978)

[2] We thank the PAAMS reviewers for their comments.