Secrecy by typing in the computational model

Stéphanie Delaune
Univ. Rennes, CNRS, IRISA, France

Clément Hérouard
Univ. Rennes, CNRS, IRISA, France

Joseph Lallemand
Univ. Rennes, CNRS, IRISA, France

***Abstract*—In this paper, we propose a way to automate proofs of cryptographic protocols in the computational setting. We focus on weak secrecy and we aim to use type systems. Techniques based on typing have been used in symbolic models, and we show how these techniques can be adapted to the CCSA framework to obtain computational guarantees.**

We only consider for now a limited set of primitives: symmetric encryption and decryption, and pairing (*i.e.* concatenation). However, our approach has the usual benefit of type systems of being modular, and could be extended to other primitives without excessive difficulties. We aim to integrate it into the SQUIRREL proof assistant so that users may show some weak secrecy properties by typing and use them as part of larger SQUIRREL developments. This is still ongoing work.

***Index Terms*—Security protocols, automated reasoning, typing, computational model, CCSA**

I. INTRODUCTION

Cryptographic protocols are distributed programs that rely on cryptographic primitives to provide security guarantees. Their design is well-known to be error-prone, which can lead to flaws with dramatic consequences as they are used in many critical applications today.

Over the past few decades, the need to ground security analysis of protocols on rigorous mathematical foundations has emerged with two distinct approaches: the *computational* one and the *symbolic* one. The symbolic approach [1] makes strong assumptions about cryptographic primitives (they are assumed to be perfect), and models messages using first-order terms. This approach benefits from automation and many procedures and tools exist [2]. In contrast, the computational approach represents messages using bit-strings [3], and agents (as well as the attacker) as probabilistic polynomial time Turing machines. The security assumptions on the primitives are expressed as *cryptographic games* (*e.g.* IND-CPA, INT-CTXT) that an attacker has a low probability of winning. This provides stronger guarantees, but makes automated reasoning much more difficult.

In order to bring these two approaches closer together, the Computationally Complete Symbolic Attacker (CCSA) logic has been proposed [4]. In this logic, messages are represented by terms, and security properties by first-order formulas. However, the semantics is a computational one, and the axioms and proof rules express computational assumptions. This approach has been implemented in the SQUIRREL proof assistant [5]–[7] and allows the user to write security proofs at a rather abstract level while obtaining computational guarantees.

This work received funding from the France 2030 program managed by the French National Research Agency under grant agreement No. ANR-22-PECY-0006

The CCSA logic focuses mainly on indistinguishability properties such as *strong secrecy* which expresses that a secret message must be indistinguishable from a randomly sampled value. In some cases, however, we need to consider the notion of *weak secrecy*, whereby an attacker cannot know the exact value of a message (except with negligible probability). For instance, a security proof may rely on an intermediate lemma stating that, at a certain point in the execution, when an agent computes $k = \mathsf{kdf}(x)$ for some message x, the same key k cannot have been derived earlier. To show that, one would have to prove that x is weakly secret. Note that the message x could have been a tag paired with a fresh nonce, and strong secrecy would not hold in this case. In contrast with indistinguishability proofs, proofs of weak secrecy tend to be rather tedious to write in SQUIRREL. In practice, one often has to instead prove strong secrecy, and deduce weak secrecy from it, which is inconvenient, and actually not always possible.

Example 1. *We consider a simple version of the Wide-Mouth Frog (WMF) protocol [8] where a server S who shares a symmetric key k_{AS} (resp. k_{BS}) with A (resp. B) acts as an intermediate to transmit a nonce n from A to B.*

$$A \to S : \quad \{n\}_{k_{AS}}$$
$$S \to B : \quad \{n\}_{k_{BS}}$$

To establish weak secrecy of n in SQUIRREL, we first prove strong secrecy of n. It takes about 80 LoC, and requires several intermediate lemmas (even for a very simple scenario).

In this paper, we propose a way to automate such proofs by using type systems. Techniques based on typing have already been successfully used in symbolic models to prove weak secrecy or authentication properties [9], [10]. We show how they can be adapted to the CCSA framework to obtain computational guarantees[1]. In short, as it is usual for such techniques, we consider types that express the level of confidentiality of the messages. We show that our typing rules ensure that from well-typed public messages, a computational attacker cannot deduce a message whose type specifies it should be secret. This allows a user to easily derive weak secrecy properties in the computational setting, by providing light type annotations and performing type-checking. We only consider for now a limited set of primitives: symmetric encryption, and pairing.

II. TYPE SYSTEM

We give here a brief overview of our type system and its associated soundness result. Many details are omitted.

[1] A recent work has independently explored a similar line of research [11].

$$\frac{\Gamma \vdash t : \mathrm{T} \qquad \Gamma(k) = \mathrm{SK[T]}}{\Gamma, r : \mathrm{Rand} \vdash \{m\}^r_k : \mathrm{L}} \qquad \frac{\Gamma \vdash c : \mathrm{L} \qquad \Gamma(k) = \mathrm{SK[T]}}{\Gamma \vdash \mathrm{sdec}(c,k) : \mathrm{T} + \mathrm{Fail}}$$

Fig. 1: Typing for symmetric encryption and decryption

A. Model

As in the CCSA logic, messages are modelled using terms. We consider the symbols $\{\cdot\}^{\cdot}_{\cdot}$, $\mathrm{sdec}(\cdot,\cdot)$, and $\mathrm{att}(\cdot)$ for symmetric encryption, decryption, and attacker computations. Terms are built using the symbols above, on top of names and variables. Each term is interpreted as a Turing machine (TM). Encryption is randomised explicitly: $\{m\}^r_k$ denotes the encryption of m with key k and random r. Moreover, we assume that the encryption and decryption functions are interpreted by algorithms satisfying the usual IND-CPA and INT-CTXT assumptions.

Example 2. *Going back to the WMF protocol, the first message sent by A is modelled as $\{n\}^{r_1}_{k_{AS}}$, whereas the one received by S is $\mathrm{att}(\{n\}^{r_1}_{k_{AS}})$ since the attacker may attempt to modify it on its way. Then, if decryption by k_{AS} succeeds, the response sent by S will be $\{\mathrm{sdec}(\mathrm{att}(\{n\}^{r_1}_{k_{AS}}), k_{AS})\}^{r_2}_{k_{BS}}$.*

B. Types and typing rules

We consider some base types: L ("Low secrecy level") for public terms, H ("High secrecy level") for secret terms, SK[T] for keys encrypting messages of type T, and Rand for randomness used in encryptions. We only allow keys to be used in key positions in terms. Our system also uses more complex types, constructed over base types (*e.g.* product and sum types), which we mostly omit here.

The type system uses an environment Γ, mapping names and variables to types. We define a judgement $\Gamma \vdash t : \mathrm{T}$, which reads "$t$ is of type T in environment Γ", by a set of typing rules. In Fig. 1, we give the rules corresponding to encryption. To encrypt a term, we check that the type of the plaintext matches that of the key, and remove the random used from the environment in order to ensure it will not be used again. Regarding the decryption rule, the type $\mathrm{T} + \mathrm{Fail}$ indicates that decryption either fails or gives a message of the type T as indicated by the type of the key.

Example 3. *Using $\Gamma := \{n : \mathrm{H}; k_{AS}, k_{BS} : \mathrm{SK[H]}; r_1, r_2 : \mathrm{Rand}\}$, messages exchanged during the protocol execution (see Example 2) type L. Derivations are written in Appendix.*

C. Soundness

To express the soundness of our system, we first give each type a computational interpretation, *i.e.*, intuitively, a set of terms whose TM interpretation can (or cannot, depending on the type) be computed from public data by a polynomial attacker. We write $\Gamma \models t : \mathrm{T}$ to denote that a term t belongs to the interpretation of type T in environment Γ. Regarding the interpretation of H, we consider an attacker who has access to an oracle that provides him with the TM interpreting any public message, *i.e.* any term of type L. We say that $\Gamma \models t : \mathrm{H}$ if such an attacker cannot compute the output of the TM interpreting t (except with negligible probability).

Our main result is the following soundness theorem.

Theorem 1. *For any term t, environment Γ, and type T, if $\Gamma \vdash t : \mathrm{T}$ then $\Gamma \models t : \mathrm{T}$.*

Proof (sketch). The main difficulty is to establish that for any term s and m such that $\Gamma \vdash s : \mathrm{H}$ and $\Gamma \vdash m : \mathrm{L}$, the interpretation of m does not help the attacker to compute s.

In order to use the IND-CPA assumption on the encryption primitive, we first have to get rid of the decryption symbols, since the IND-CPA game does not allow the attacker to decrypt messages. We proceed in two steps.

1) We show that, if $\Gamma \vdash m : \mathrm{L}$, then there exists another term m', whose TM interpretation is the same as m (with overwhelming probability), and which can be typed with type L in a restricted fragment of the type system that excludes in particular the decryption rule. This step relies on the INT-CTXT assumption which states that any ciphertext that successfully decrypts has been produced by the oracle, meaning that the resulting plaintext is equal to a message encrypted in a public term.
2) We establish a soundness result for the restricted fragment using the assumptions on the cryptographic primitives. In the case of encryption, consider the case where m' contains a ciphertext typed by the rule shown in Fig. 1. We construct an attacker against IND-CPA, whose advantage is at least the probability to compute s from m'.

This concludes the proof for symmetric encryption. □

As a corollary of Theorem 1, we deduce that if the messages exchanged during the protocol execution are typed L, then this is also the case for any computation performed by the attacker using these messages, and therefore the result can not be equal to a term s having type H: we have that s is weakly secret.

Example 4. *We saw in Example 3 that messages exchanged during the protocol execution are typed L in environment Γ where $n : \mathrm{H}$. By Theorem 1, we have $\Gamma \models n : \mathrm{H}$. So n is weakly secret, i.e. the attacker can only compute its interpretation with negligible probability from the messages it observed.*

III. Work in progress

We are currently working on generalising our typing-based approach in several ways. First, applying our result currently requires to show that any message that can be output by the protocol is indeed of type L. SQUIRREL nicely represents the possible executions of protocols, including input and output messages, as well as mutable states, using so-called *macros*. These are, roughly, recursively defined functions in the CCSA logic [12]. To be able to incorporate our type system into SQUIRREL, we need to adapt it to handle these macros. The idea is to build an inductive typing proof, in which we may type messages assuming inputs are public, because the attacker must have computed them using previous outputs, which were typed as public. Recursive functions in CCSA provide a framework for a well-funded order to build such inductive proofs. Second,

we are also working on adding other standard cryptographic primitives. While the work described here focuses on symmetric encryption, we plan to add support for IND-CCA asymmetric encryption and PRF keyed hash functions. In addition, our end goal is to implement a type-checker for our type system into the SQUIRREL prover, so that users may show weak secrecy properties by typing, and use them as part of larger SQUIRREL developments.

REFERENCES

[1] D. Dolev and A. C. Yao, "On the security of public key protocols," *IEEE Trans. Information Theory*, vol. 29, no. 2, pp. 198–207, 1983.

[2] M. Barbosa, G. Barthe, K. Bhargavan, B. Blanchet, C. Cremers, K. Liao, and B. Parno, "SoK: Computer-aided cryptography," in *2021 2021 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2021, pp. 777–795. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP40001.2021.00008

[3] S. Goldwasser and S. Micali, "Probabilistic encryption," *J. Comput. Syst. Sci.*, vol. 28, no. 2, pp. 270–299, 1984. [Online]. Available: https://doi.org/10.1016/0022-0000(84)90070-9

[4] G. Bana and H. Comon-Lundh, "A computationally complete symbolic attacker for equivalence properties," in *CCS*. ACM, 2014, pp. 609–620.

[5] D. Baelde, S. Delaune, C. Jacomme, A. Koutsos, and S. Moreau, "An interactive prover for protocol verification in the computational model," in *IEEE Symposium on Security and Privacy*. IEEE, 2021, pp. 537–554.

[6] D. Baelde, S. Delaune, A. Koutsos, and S. Moreau, "Cracking the stateful nut: Computational proofs of stateful security protocols using the squirrel proof assistant," in *CSF*. IEEE, 2022, pp. 289–304.

[7] The Squirrel Prover repository. https://github.com/squirrel-prover/squirrel-prover/.

[8] M. Burrows, M. Abadi, and R. M. Needham, "A logic of authentication," *ACM Trans. Comput. Syst.*, vol. 8, no. 1, pp. 18–36, 1990. [Online]. Available: https://doi.org/10.1145/77648.77649

[9] M. Bugliesi, R. Focardi, and M. Maffei, "Authenticity by tagging and typing," in *2004 ACM Workshop on Formal Methods in Security Engineering*, ser. FMSE '04. New York, NY, USA: ACM, 2004, pp. 1–12.

[10] R. Focardi and M. Maffei, "Types for security protocols," in *Formal Models and Techniques for Analyzing Security Protocols*, ser. Cryptology and Information Security Series. IOS Press, 2011, vol. 5, ch. 7, pp. 143–181.

[11] J. Gancher, S. Gibson, P. Singh, S. Dharanikota, and B. Parno, "Owl: Compositional verification of security protocols via an information-flow type system," in *IEEE Symposium on Security and Privacy*. IEEE, 2023.

[12] D. Baelde, A. Koutsos, and J. Lallemand, "A higher-order indistinguishability logic for cryptographic reasoning," in *38th Symposium on Logic in Computer Science (LICS)*, 2023.

APPENDIX

To represent formally the messages from the WMF protocol, we expand our signature with:

- a function of arity 3 expressing conditional branching "if $\cdot$ then $\cdot$ else $\cdot$",
- a constant fail representing the value returned when the decryption algorithm fails,
- and a constant empty representing an empty message.

The first message sent by A is $m_1 = \{n\}^{r_1}_{k_{AS}}$. Then, we denote by t the result of the decryption performed by the server S: *i.e.* $t = \mathrm{sdec}(\mathrm{att}(m_1), k_{AS})$. Lastly, the message sent by S and denoted m_2 is:

$$m_2 = \text{if } t \neq \text{fail then } \{t\}^{r_2}_{k_{BS}} \text{ else empty}.$$

In order to type m_1 and m_2, we need to introduce two other types: Bool for boolean terms and Fail for the term fail. When we derive typing judgements, we have to make sure that the randomness used for encryption are used only once in order to ensure the soundness of our typing rules. Formally, an environment Γ is *well-formed*, denoted $\Gamma \vdash \diamond$, if each variable is bound at most once, and we write $\Gamma \to (\Gamma_1, ..., \Gamma_n)$ when

$$\{r \mid r : \mathrm{Rand} \in \Gamma\} = \begin{array}{ll} & \{r \mid r : \mathrm{Rand} \in \Gamma_1\} \\ \uplus & \dots \\ \uplus & \{r \mid r : \mathrm{Rand} \in \Gamma_n\}. \end{array}$$

Here $\uplus$ represents disjoint union.

In addition to the two rules (SDEC/SENC) given in Figure 1, we present some extra rules in Figure 3. These rules are the most important ones, and allow us to present the derivations for the WMF protocol.

We will use these rules to type messages m_1 and m_2 learned by the attacker during the execution of the WMF protocol. We consider the environment

$$\Gamma := \{n : \mathrm{H};\ k_{AS} : \mathrm{SK[H]};\ k_{BS} : \mathrm{SK[H]}\}.$$

The typing derivation for m_1 is given in Figure 2. It uses the SENC rule and removes the random r_1 from the environment.

$$\dfrac{\dfrac{\Gamma \vdash \diamond}{\Gamma \vdash n : \mathrm{H}}\ (\textsc{Env}) \qquad \Gamma(k_{AS}) = \mathrm{SK[H]}}{\Gamma;\ r_1 : \mathrm{Rand} \vdash \{n\}^{r_1}_{k_{AS}} : \mathrm{L}}\ (\textsc{Senc})$$

Fig. 2: Typing derivation D_1 for message m_1

The typing derivation for m_2, shown in Figure 4, illustrates the use of the rule ASSIGN. The random r_1 appears twice in m_2. As we must use it only once in our derivation, we will introduce a variable x to store this encryption. We define m'_2 as the message m_2 in which we replaced both occurrences of t by the variable x. This way, the random r_1 does not appear in m'_2, and $m'_2[x \mapsto t] = m_2$. The term m'_2 will be typed in two different ways depending on whether the decryption succeeds (type H) or fails (type Fail). We rely on the rule BREAKSUM on the variable x to handle these two cases.

The derivation in Figure 4 reuses D_1 written in Figure 2, and refers to derivations D_2 and D_3 shown in Figures 5 and 6. We omit in D_2 the straightforward derivation that $x \neq \text{fail}$ has type Bool.

The two cases derived by the rule BREAKSUM illustrate two different ways to handle conditions. When x is of type H (Figure 5), both branches of the conditional can be typed with the rule IF. However, when x is of type Fail (Figure 6), the term $\{x\}^{r_2}_{k_{BS}}$ cannot be typed. Indeed, the type of x does not match the type expected for a key having type SK[H]. So, we use the rule IFFAIL to ignore this branch. The typing ensures that this conditional is indeed always false.

In the end, we have shown that m_1 and m_2 have a public type (L) in environment Γ, $r_1 : \mathrm{Rand}$, $r_2 : \mathrm{Rand}$.

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash n : \Gamma(n)}\ (\textsc{Env})$$

$$\frac{\Gamma \vdash \diamond}{\Gamma \vdash \text{empty} : \mathrm{L}}\ (\textsc{Empty}) \qquad \frac{\Gamma \vdash t : \mathrm{L}}{\Gamma \vdash \text{att}(t) : \mathrm{L}}\ (\textsc{Att})$$

$$\frac{\Gamma_1, x : \mathrm{T}' \vdash t : \mathrm{T} \qquad \Gamma_2 \vdash t' : \mathrm{T}' \qquad \Gamma \to (\Gamma_1, \Gamma_2)}{\Gamma \vdash t[x \mapsto t'] : \mathrm{T}}\ (\textsc{Assign})$$

$$\frac{\Gamma, x : \mathrm{T}_1 \vdash t : \mathrm{T} \qquad \Gamma, x : \mathrm{T}_2 \vdash t : \mathrm{T}}{\Gamma, x : \mathrm{T}_1 + \mathrm{T}_2 \vdash t : \mathrm{T}}\ (\textsc{BreakSum})$$

$$\frac{\Gamma_1 \vdash t_1 : \mathrm{Bool} \qquad \Gamma_2 \vdash t_2 : \mathrm{T} \qquad \Gamma_3 \vdash t_3 : \mathrm{T} \qquad \Gamma \to (\Gamma_1, \Gamma_2, \Gamma_3)}{\Gamma \vdash \text{if } t_1 \text{ then } t_2 \text{ else } t_3 : \mathrm{T}}\ (\textsc{If})$$

$$\frac{\Gamma_1 \vdash t_1 : \mathrm{Fail} \qquad \Gamma_3 \vdash t_3 : \mathrm{T} \qquad \Gamma \to (\Gamma_1, \Gamma_3)}{\Gamma \vdash \text{if } t_1 \neq \text{fail then } t_2 \text{ else } t_3 : \mathrm{T}}\ (\textsc{IfFail})$$

Fig. 3: Typing rules

$$\dfrac{\dfrac{\dfrac{D_2}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{H} \vdash m_2' : \mathrm{L}} \qquad \dfrac{D_3}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{Fail} \vdash m_2' : \mathrm{L}}\ (\textsc{BreakSum})}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{H} + \mathrm{Fail} \vdash m_2' : \mathrm{L}} \qquad \dfrac{\dfrac{\dfrac{D_1}{\Gamma, r_1 : \mathrm{Rand} \vdash m_1 : \mathrm{L}}}{\Gamma, r_1 : \mathrm{Rand} \vdash \text{att}(m_1) : \mathrm{L}}\ (\textsc{Att}) \qquad \Gamma(k_{AS}) = \mathrm{SK[H]}}{\Gamma, r_1 : \mathrm{Rand} \vdash t : \mathrm{H} + \mathrm{Fail}}}{\Gamma, r_1 : \mathrm{Rand}, r_2 : \mathrm{Rand} \vdash m_2 : \mathrm{L}}\ (\textsc{Assign})$$

Fig. 4: Typing derivation for message m_2

$$\dfrac{\dfrac{\dots}{\Gamma, x : \mathrm{H} \vdash x \neq \text{fail} : \mathrm{Bool}} \qquad \dfrac{\dfrac{\Gamma, x : \mathrm{H} \vdash \diamond}{\Gamma, x : \mathrm{H} \vdash x : \mathrm{H}}\ (\textsc{Env}) \qquad \Gamma(k_{BS}) = \mathrm{SK[H]}}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{H} \vdash \{x\}^{r_2}_{k_{BS}} : \mathrm{L}}\ (\textsc{Senc}) \qquad \dfrac{\Gamma, x : \mathrm{H} \vdash \diamond}{\Gamma, x : \mathrm{H} \vdash \text{empty} : \mathrm{L}}\ (\textsc{Empty})}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{H} \vdash \text{if } x \neq \text{fail then } \{x\}^{r_2}_{k_{BS}} \text{ else empty} : \mathrm{L}}\ (\textsc{If})$$

Fig. 5: Typing derivation D_2 - case x of type H

$$\dfrac{\dfrac{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{Fail} \vdash \diamond}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{Fail} \vdash x : \mathrm{Fail}}\ (\textsc{Env}) \qquad \dfrac{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{Fail} \vdash \diamond}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{Fail} \vdash \text{empty} : \mathrm{L}}\ (\textsc{Empty})}{\Gamma, r_2 : \mathrm{Rand}, x : \mathrm{Fail} \vdash \text{if } x \neq \text{fail then } \{x\}^{r_2}_{k_{BS}} \text{ else empty} : \mathrm{L}}\ (\textsc{IfFail})$$

Fig. 6: Typing derivation D_3 - case x of type Fail