A Probabilistic Hoare-style logic for Game-based Cryptographic Proofs

Ricardo Corin and Jerry den Hartog

{ricardo.corin,jerry.denhartog}@cs.utwente.nl
Department of Computer Science, University of Twente, The Netherlands

Abstract. We extend a Probabilistic Hoare-style logic to formalize game-based cryptographic proofs. Our approach provides a systematic and rigorous framework, thus preventing errors from being introduced. We illustrate our technique by proving semantic security of ElGamal.

1 Introduction

A typical proof to show that a cryptographic construction is secure uses a reduction from the desired security notion towards some underlying hardness assumption. The security notion is usually represented as a game, in which one proves that the attacker's chance of winning the game is (arbitrarily) small. From a programming language perspective, these games can be thought of as programs whose behaviour is partially known, since the program typically contains invocations to an unknown function representing an arbitrary attacker. In this context, the cryptographic reduction is a sequence of valid program transformations.

Even though cryptographic proofs based on game reductions are powerful, the price one has to pay is high: these proofs are complex, and can easily become involved and intricated. This makes the verification difficult, with subtle errors difficult to spot. Some errors may remain uncovered long after publication, as illustrated for example by Boneh and Franklin's IBE encryption scheme [4], whose cryptographic proof has been recently patched by Galindo [8].

Recently, several papers from the cryptographic community (e.g., the work of Bellare and Rogaway [2], Halevi [9], and Shoup [16]) have recognized the need to tame the complexity of cryptographic proofs. There, the need for (development of) rigorous tools to organize cryptographic proofs in a systematic way is advocated. These tools would prevent subtle easily overlooked mistakes from being introduced in the proof. As another advantage, this precise proof development framework would standardize the proof writing language so that proofs can be checked easily, even perhaps using computer aided verification.

The proposed frameworks [2, 9, 16] provide ad hoc formalisms to reason about the sequences of games, providing useful program transformation rules and illustrating the techniques with several cryptographic proofs from the literature. As we mentioned earlier, the games may be thought of as computer programs, and the reductions thought of as valid program transformations, i.e. transformations that do not change (significantly at least) the "behaviour" of the program. If

we represent program behaviour by predicates that establish which states are satisfied by the program before and after its execution, we arrive to a well known setting studied by computer scientists for the past thirty years: program correctness established by a Hoare logic [12].In Hoare logic, a programming language statement (e.g., value assignment to a variable) is prefixed and postfixed with assertions which state which conditions hold before and after the execution of the statement, respectively. There exists a wealth of papers building on the basic Hoare logic setting, making it one of the most studied subjects for establishing (imperative) program correctness.

This paper's contributions are twofold. First, we adapt and extend our earlier work on Probabilistic Hoare-logic [10, 11] to cope with game-based cryptographic proofs. In particular, we introduce the notion of arbitrary functions, that can be used to model the invocation of an unknown computation (e.g., an arbitrary attacker function). We also include procedures, which are subroutines that can be used to "wrap" function invocations. We provide the associated deduction rules within the logic.We also present a useful program transformation operation, called *orthogonality*, which we use to relate Hoare triples. Orthogonality is our basic "game stepping" operation. Second, to illustrate our approach, we elaborate in full detail a proof of security of ElGamal [6], by reducing the semantic security of the cryptosystem to the hardness of solving the (well-known) Decisional Diffie-Hellman problem.

To the best of our knowledge, ours is the first application of a well known program correctness logic (i.e. Hoare logic) to analyze cryptographic proofs based on transformation of probabilistic imperative programs.

A longer version of this paper [5] contains additional details and proofs.

Related Work Differently from us, almost all formalisms we know of are directed towards analysing security protocols, thus including concurrency as a main modelling operation. One prime example is in the work of Ramanathan et al. [15], where a probabilistic poly-time process algebraic language is presented. Much effort is paid to measure the computational power of (possibly parallel) processes, so that an environmental context can be precisely regulated to run in probabilistic polynomial time. On the other hand, our logic is fitted for proofs on a simple probabilistic imperative language, without considering parallel systems, nor communication or composition. This simplifies the reasoning and is closer to the original cryptographic proofs which always consider imperative programs (the "games").

Tarento [17] develops machine checkable proofs of signature schemes, focusing on formalizing the semantics of the generic and random oracle models. This differs from the present work, which uses a Hoare-style logic to "derive" the (syntactic) cryptographic algorithms, and then uses the soundness of the logic to obtain the security proofs.

Recently, Blanchet [3] has developed an automated procedure to generate security proofs of protocols; the approach is similar to ours in that also sequences of games are used, although our technique, based on Hoare-logic derivations, can be

used to develop proofs manually (however proof checking could be automated); still, it would be interesting to relate the approaches in the future.

2 The Probabilistic Hoare-style logic *pL*

We shortly recall the probabilistic Hoare style logic pL (see [10, 11]). We introduce probabilistic states, and programs which transform such states. Then we introduce probabilistic predicates and a reasoning system to establish Hoare triples which link a precondition and a postcondition to a program.

Probabilistic programs We define programs (or statements) s, integer expressions e and Boolean expressions (or conditions) c by:

$$s ::= \texttt{skip} \mid x \texttt{ := } e \mid s \texttt{ ; } s \mid \texttt{if } c \texttt{ then } s \texttt{ else } s \texttt{ fi} \mid \texttt{while } c \texttt{ do } s \texttt{ od} \mid s \oplus_\rho s$$
$$e ::= n \mid x \mid e + e \mid e - e \mid e \cdot e \mid e \texttt{ div } e \mid e \texttt{ mod } e$$
$$c ::= \texttt{true} \mid \texttt{false} \mid b \mid e \texttt{ = } e \mid e < e \mid c \wedge c \mid c \vee c \mid \neg c \mid c \to c$$

where x is a variable of *type* (or 'has *range*') integer, b is a variable of type Boolean and n a number. We assume it is clear how this can be extended with additional operators and to other types and mostly leave the type of variables implicit, assuming that all variables and values are of the correct type.

The basic statements do nothing (`skip`) and assignment (x `:=` e) can be combined with sequential composition (`;`), conditional choice (`if`), iteration (`while`) and probabilistic choice $\oplus_\rho$. In the statement $s \oplus_\rho s'$ a probabilistic decision is made which results in executing s with probability ρ and statement s' with probability $1 - \rho$.

A *deterministic state*, $\sigma \in \mathcal{S}$, is a function that maps each program variable to a value. A *probabilistic state*, $\theta \in \Theta$ gives the probability of being in a given deterministic state. Thus a probabilistic state θ can be seen as a (countable) weighed set of deterministic states which we write as $\rho_1 \cdot \sigma_1 + \rho_2 \cdot \sigma_2 + \ldots$. Here, the probability of being in the (deterministic) state σ_i is ρ_i, $i \geq 0$. For simplicity and without loss of generality we assume that each state σ occurs at most once in θ; multiple occurrences of a single state can be merged into one single occurrence by adding the probabilities, e.g. $1 \cdot \sigma$ rather than $\frac{3}{4} \cdot \sigma + \frac{1}{4} \cdot \sigma$.

The sum of all probabilities is at most 1 but may be less. A probability less than 1 indicates that this execution point may not be always reached (e.g., because of non-termination or because it is part of an '`if`' conditional branch).

To manipulate and combine states we have *scaling* ($\rho \cdot \theta$) which scales the probability of each state in θ, *addition* ($\theta + \theta'$) which unites the two sets and adds probabilities if the same state occurs in both θ and θ', *weighed sum* ($\theta \oplus_\rho \theta' = \rho \cdot \theta + (1-\rho) \cdot \theta'$) and *conditional selection* ($c?\theta$) which selects the states satisfying c (and removes the rest). For example,

$$\tfrac{1}{2} \cdot (\tfrac{1}{2} \cdot [x = 1] + \tfrac{1}{2} \cdot [x = 2]) = \tfrac{1}{4} \cdot [x = 1] + \tfrac{1}{4} \cdot [x = 2]$$
$$(\tfrac{1}{4} \cdot [x = 1] + \tfrac{1}{4} \cdot [x = 2]) + \tfrac{1}{4} \cdot [x = 2] = \tfrac{1}{4} \cdot [x = 1] + \tfrac{1}{2} \cdot [x = 2]$$
$$(x \leq 2)?(\tfrac{1}{4} \cdot [x = 1] + \tfrac{1}{2} \cdot [x = 2] + \tfrac{1}{4} \cdot [x = 3]) = \tfrac{1}{4} \cdot [x = 1] + \tfrac{1}{2} \cdot [x = 2]$$

A program s is interpreted as a transformer of probabilistic states, i.e. its *semantics* $\mathcal{D}(s)$ is a function that maps input states of s to output states. The program transforms the probabilistic state element-wise, with the usual interpretation of the deterministic operations. (See [10] for the fixed point construction used for the semantics of `while`.) For probabilistic choice we use the weighed sum: $\mathcal{D}(s \oplus_\rho s')(\theta) = \mathcal{D}(s)(\theta) \oplus_\rho \mathcal{D}(s')(\theta)$.

Reasoning about probabilistic programs To reason about deterministic states we use *deterministic predicates*, $dp \in DPred$. These are first order logical formulas, i.e. Boolean expressions with the addition of logical variables i, j and the quantification $\forall i :, \exists i :$ over such variables. Similarly, to reason about probabilistic states and programs we introduce *probabilistic predicates*, $p \in Pred$:

$$\begin{aligned} p ::= {} & \texttt{true} \mid \texttt{false} \mid b \mid e = e \mid e < e \mid e_r = e_r \mid e_r < e_r \mid p \to p \mid \neg p \\ & \mid p \wedge p \mid p \vee p \mid \exists j : p \mid \forall j : p \mid \rho \cdot p \mid p + p \mid p \oplus_\rho p \mid c?p \\ e_r ::= {} & \rho \mid \texttt{r} \mid \mathbb{P}(dp) \mid e_r + e_r \mid e_r - e_r \mid e_r * e_r \mid e_r / e_r \mid \ldots \end{aligned}$$

where e is an expression using logical variables rather than program variables, ρ is a real number and $\texttt{r}$ a variable with range $[0, 1]$. A *probabilistic expression* e_r is meant to express a probability in $[0, 1]$.

Example 1. We have that $(i < j) \to (\mathbb{P}(x = 5 \wedge y < x + i) > \mathbb{P}(x = j) + \frac{1}{4})$ is a probabilistic predicate but $(x > i)$ is not as the use of program variable x outside of the $\mathbb{P}(\cdot)$ construction is not allowed.

The value of $\mathbb{P}(dp)$, in a given probabilistic state, is the sum of the probabilities for deterministic states that satisfy dp, e.g. in $\frac{1}{4} \cdot [x = 1] + \frac{1}{4} \cdot [x = 2] + \frac{1}{4} \cdot [x = 3] + \frac{1}{4} \cdot [x = 4]$ we have that $\mathbb{P}(x \geq 2) = \frac{3}{4}$. Establishing the value of a probabilistic expression e_r and a (basic) predicate p from a probabilistic state θ is standard; the latter is denoted (as usual) by the satisfaction relation $\theta \models p$. The 'arithmetical' operators $+, \oplus_\rho, \rho\cdot, ?$ specific to our probabilistic logic are the logical counterparts of the same operations on states. For example,

$$\theta \models p + p' \text{ when there exists } \theta_1, \theta_2 \text{: } \theta = \theta_1 + \theta_2, \theta_1 \models p \text{ and } \theta_2 \models p' \tag{1}$$

$$\theta \models c?p \text{ when there exists } \theta' \text{: } \theta = c?\theta', \theta \models p \tag{2}$$

The satisfaction relation also includes an interpretation function giving values to the logical variables, which we omit from the notation when no confusion is possible. We write $\models p$ if p holds in any probabilistic state.

Hoare triples, also known as *program correctness triples*, give a precondition and a postcondition for a program. A triple is called valid, denoted $\models \{\, p \,\}\, s \,\{\, q \,\}$, if the precondition guarantees the postcondition after execution of the program, i.e. for all θ with $\theta \models p$ we have $\mathcal{D}(s)(\theta) \models q$.

Our derivation system for Hoare triples adapts and extends the existing Hoare logic calculus. The standard rules for skip, assignment, sequential composition, precondition strengthening and postcondition weakening remain the same. The rule for conditional choice is adjusted and a new rule for probabilistic choice is added, along with some structural rules. We only present the main rules here

(see e.g. [10] for a complete overview), noting that the other rules come directly from Hoare logic or from natural deduction.

$$\{\, p[x/e] \,\}\ x \texttt{ := } e\ \{\, p \,\} \quad \text{(Assign)} \qquad \frac{\{\, c?p \,\}\ s\ \{\, q \,\} \quad \{\, \neg c?p \,\}\ s'\ \{\, q' \,\}}{\{\, p \,\}\ \texttt{if}\ c\ \texttt{then}\ s\ \texttt{else}\ s'\ \texttt{fi}\ \{\, q + q' \,\}}\ \text{(If)}$$

$$\frac{\{\, p \,\}\ s\ \{\, p' \,\} \quad \{\, p' \,\}\ s'\ \{\, q \,\}}{\{\, p \,\}\ s\,;\,s'\ \{\, q \,\}}\ \text{(Seq)} \qquad \frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p \,\}\ s'\ \{\, q' \,\}}{\{\, p \,\}\ s \oplus_\rho s'\ \{\, q \oplus_\rho q' \,\}}\ \text{(Prob)}$$

$$\frac{\{\, p \,\}\ s\ \{\, q \,\} \quad \{\, p \,\}\ s\ \{\, q' \,\}}{\{\, p \,\}\ s\ \{\, q \wedge q' \,\}}\ \text{(And)} \qquad \frac{\models p' \to p \quad \{\, p \,\}\ s\ \{\, q \,\} \quad \models q \to q'}{\{\, p' \,\}\ s\ \{\, q' \,\}}\ \text{(Cons)}$$

These rules are used in the proof of ElGamal in Section 4, but first we extend the language and logic to cover the necessary elements for cryptographic proofs.

3 Extending *pL*

We consider two language extensions and one extension of the reasoning method:

- *Functions* are computations that are a priori unknown. These are useful to reason about arbitrary *attacker* functions, for which we do not know what behavior they will produce.
- *Procedures* allow the specification of subroutines. These are useful to specify cryptographic assumptions that hold 'for every procedure' satisfying some appropriate conditions. Procedures are programs for which its behavior (i.e. the procedure's *body*) is assumed to be partially known (since it may contain an invocation to an arbitrary function).
  We assume that both functions and procedures are deterministic. However this poses no loss of generality as enough "randomness" can be sampled before and then passed to the function or procedure as an extra parameter. We explicitly distinguish functions and procedures for readability and convenience, rather than because there is a fundamental difference between the two; it clarifies the different roles (i.e. procedures are specified routines and functions are unknown attacker functions) directly in the syntax.
- *Orthogonality* allows to reason about independent statements. This is a program transformation operation that is going to be useful when reasoning on cryptographic proofs as sequences of games.

Functions Functions, as opposed to procedures, are undefined (i.e. we do not provide a body). We use these functions to represent arbitrary attackers, for which we do not know a priori their behaviour.

To include functions in the language we add function symbols to expressions (as defined in the previous section): $e ::= \ldots \mid f(e, \ldots, e)$. We assume that the

functions are used correctly, that is functions are always invoked with the right number of arguments and correct types. Also, note that by considering functions to be expressions we allow functions to be used in the (deterministic and probabilistic) predicates. The fact that a function is deterministic is represented in the logic by the following remark.

Remark 2. For any function $f(\cdot)$ (of arity n) and expressions $e_1, \ldots, e_n, e'_1, \ldots, e'_n$ we have $\models (e_1 = e'_1 \wedge \ldots \wedge e_n = e'_n) \rightarrow f(e_1, \ldots, e_n) = f(e'_1, \ldots, e'_n)$.

To deal with functions in the semantics, we assume that any function symbol f has some fixed (albeit unknown) deterministic, type correct interpretation $\hat{f}$. Thus, e.g. the semantics for an assignment using f becomes $\mathcal{D}(x := f(y))(\rho_1 \cdot \sigma_1 + \rho_2 \cdot \sigma_2 + \ldots) = \rho_1 \cdot \sigma_1[^{\hat{f}(\sigma_1(\mathtt{y}))} / _{\mathtt{x}}] + \rho_2 \cdot \sigma_2[^{\hat{f}(\sigma_2(\mathtt{y}))} / _{\mathtt{x}}] + \ldots$. The rules given above are also valid for the extended language; extending the correctness proof [10] for the Assign rule is direct, while the proof for the other rules remains the same as it only uses structural properties of the denotational semantics.

Procedures We now extend the language with procedures, which are used to model (partially) known subprograms. Each procedure has a list of variables, the *formal parameters* (divided in turn into value parameters and variable parameters) and a set of local variables. We assume that none of these variables occur in the main program or in other procedures. The procedure also has a body, $\mathtt{B}_{proc}$, which is a program statement which uses only the formal parameters and local variables, only assigns to variable parameters and local variables, and assigns to a local variable before using its value. We also enforce the procedure to be deterministic by excluding any probabilistic choice statement from $\mathtt{B}_{proc}$. Finally, we require that the procedure is non-recursive (i.e. we can order procedures such that any procedure only calls procedures of a lower order). We use the notation procedure $proc(\textbf{value } \mathtt{v}_1, \ldots, \mathtt{v}_n; \textbf{var } \mathtt{w}_1, \ldots, \mathtt{w}_m) : \mathtt{B}_{proc}$ to list the value and variable parameters and the body of a procedure (any variables in $\mathtt{B}_{proc}$ that are not formal parameters are local variables).

We add procedures to the language by including *procedure* calls to the statements, $s ::= \ldots \mid proc(e, \ldots, e; x, \ldots, x)$. Here we assume that there is *no aliasing of variables*; i.e. a different variable is used for each variable parameter.

The procedure call $proc(e_1, \ldots, e_n, x_1, \ldots, x_m)$ (in state σ) corresponds to first assigning the value of the appropriate expression (e_i or x_j) to the formal parameters, running the body of the program and finally assigning the resulting value of the variable arguments $\mathtt{w}_1, \ldots, \mathtt{w}_m$ to $x_1, \ldots, x_m$. Thus the semantics is:

$$\begin{aligned}\mathcal{D}(proc(e_1, \ldots, e_n; x_1, \ldots, x_m))(\theta) = \mathcal{D}(&\mathtt{v}_1 := e_1; \ldots; \mathtt{v}_n := e_n;\\ &\mathtt{w}_1 := x_1; \ldots; \mathtt{w}_m := x_m;\\ &\mathtt{B}_{proc};\ x_1 := \mathtt{w}_1; \ldots; x_n := \mathtt{w}_n)(\theta)\end{aligned}$$

To enable reasoning about a procedure $proc(\textbf{value } \mathtt{v}_1, \ldots, \mathtt{v}_n; \textbf{var } \mathtt{w}_1, \ldots, \mathtt{w}_m) : \mathtt{B}_{proc}$, we add the following derivation rule:

$$\frac{\{p\}\ \mathtt{B}_{proc}\ \{q\}}{\{p[^{\mathtt{e}_1,\ldots,\mathtt{e}_n,\mathtt{x}_1,\ldots,\mathtt{x}_n} /_{\mathtt{v}_1,\ldots,\mathtt{v}_n,\mathtt{w}_1,\ldots,\mathtt{w}_m}]\}\ proc(\mathtt{e}_1, \ldots, \mathtt{e}_n; \mathtt{x}_1, \ldots, \mathtt{x}_n)\ \{q[^{\mathtt{x}_1,\ldots,\mathtt{x}_n} /_{\mathtt{w}_1,\ldots,\mathtt{w}_m}]\}} \quad (3)$$

The extended logic including this rule is correct, i.e. any Hoare triple derived from the proof system is valid. Extending the correctness proof for the added rule is again a simple exercise using the definition of the semantics given above and properties of the assignment statement.

Distributions and independence We now illustrate how to express the (joint) distribution of variables (and more generally of expressions) in the logic. Then we discuss the issue of independence of variables and expressions.

A commonly used component in (security) games is a variable chosen completely at *random*, which in other words is a variable with a *uniform distribution* over its (finite) range. Suppose that variable x and i have the same range S. Then the following predicate expresses that x is uniformly distributed over S:

$$R_S(x) \;=\; \forall i : \mathbb{P}(\mathtt{x} = i) = 1/|S|$$

where $|S|$ denotes the size of the set S. The variable x can be given a uniform distribution over $S = \{v_1, \ldots, v_n\}$ by running the program

$$\mathtt{x} := \mathtt{v}_1 \oplus_{1/\mathtt{n}} (x := \mathtt{v}_2 \oplus_{1/(\mathtt{n}-1)} (\cdots \oplus_{1/2} x := \mathtt{v}_\mathtt{n}))$$

As this is a commonly used construction we introduce a shorthand notation for this statement: $\mathtt{x} \leftarrow S$. Using our logic, it is straightforward to derive (using repeatedly rule (Prob)) that after running this program x has a uniform distribution over S: $\models \{\, \mathbb{P}(\mathtt{true}) = 1 \,\}\ x \leftarrow S\ \{\, R_S(x) \,\}$.

More interestingly, after running the program $\mathtt{x} \leftarrow S; \mathtt{y} \leftarrow S'$ we not only know that $\mathtt{x}$ has a uniform distribution over S and $\mathtt{y}$ has a uniform distribution over S', but we also know that $\mathtt{y}$ has a uniform distribution over S' *independently* from the value of $\mathtt{x}$. In other words, the *joint distribution* of $\mathtt{x}$ and $\mathtt{y}$ is $R_{S,S'}(x,y) \;::=\; \forall i, j : \mathbb{P}(x = i \wedge y = j) = 1/|S| \cdot 1/|S'|$ (with $i \in S$, $j \in S'$). This is a stronger property than only the information that x and y are uniformly distributed. (The difference is exactly the independence of the variables.) Below we introduce a predicate expressing independence and generalize these results.

Definition 3 (Independent $I(\cdot)$ and Random $R(\cdot)$ expressions). *The predicate $I(e_1, \ldots, e_n)$ states independence of expressions $e_1, \ldots, e_n$, and is defined by (where i_j is of the same type as e_j, $1 \leq j \leq n$.):*

$$I(e_1, \ldots, e_n) \;=\; \forall i_1, \ldots, i_n : \mathbb{P}(e_1 = i_1 \wedge \ldots \wedge e_n = i_n) = \mathbb{P}(e_1 = i_1) \cdot \ldots \cdot \mathbb{P}(e_n = i_n)$$

The predicate $R_{S_1,\ldots,S_n}(e_1, \ldots, e_n)$ states that $e_1, \ldots, e_n$ are randomly and independently distributed over $S_1, \ldots, S_n$ respectively, is defined as follows:

$$R_{S_1,\ldots,S_n}(e_1, \ldots, e_n) \;=\; \forall i_1, \ldots, i_n : \mathbb{P}(e_1 = i_1 \wedge \ldots \wedge e_n = i_n) = 1/|S_1| \cdot \ldots \cdot 1/|S_n|$$

Lemma 4 (Relations between $R(\cdot)$ and $I(\cdot)$).

1. *An expression list has a joint uniform distribution when they are independent and each has a uniform distribution, i.e.* $\models R_{S_1,\ldots,S_n}(e_1, \ldots, e_n) \leftrightarrow R_{S_1}(e_1) \wedge \ldots \wedge R_{S_n}(e_n) \wedge I(e_1, \ldots, e_n)$.

2. *Separate randomly assigned variables have a joint random distribution:* $\models \{\, \mathbb{P}(\mathtt{true}) = 1 \,\}\ \mathtt{x}_1 \leftarrow S_1; \ldots; \mathtt{x}_n \leftarrow S_n\ \{\, R_{S_1,\ldots,S_n}(x_1, \ldots, x_n) \,\}$.
3. *Independence is maintained by functions; if an expression* e *is independent from the inputs* $e_1, \ldots, e_n$ *of a function* f*, then* e *is also independent of* $f(e_1, \ldots, e_n)$*, i.e.,* $\models I(e, e_1, \ldots, e_n) \rightarrow I(e, f(e_1, \ldots, e_n))$.

Both (1) and (3) express basic properties, shown easily to hold semantically for any (probabilistic) state. The triple in (2) is shown valid by using the logic.

Example 5. The lemma above can be used in a derivation as follows:

$$
\begin{array}{l}
\{\mathbb{P}(\mathtt{true}) = 1\} \\
\quad b \leftarrow \mathit{Bool}\ ; \\
\{R_{\mathit{Bool}}(b)\} \\
\quad x \leftarrow S; \\
\{R_{\mathit{Bool},S}(b, x)\} \rightarrow \{I(b, x)\} \rightarrow \{I(b, f(x))\} \\
\quad b' := f(x)\ ; \\
\{I(b, b')\}
\end{array}
$$

The derivation above is represented as a so called *proof outline*, which is a commonly used way to represent proofs in Hoare logic. Briefly, rather than giving a complete proof tree only the most relevant steps of the proof are given in an intuitively clear format. The predicates in between the program statements give properties that are valid at that point in the execution.

Orthogonality A (terminating) program that does not change the value of variables in a predicate (i.e. is 'orthogonal to the predicate') will not change its truth value. In this section we make this intuitive property more precise. As we show in the proof of ElGamal cryptosystem in Section 4, orthogonality is a powerful method to reason about programs and Hoare triples yet is easy to use as it only requires a simple syntactical check.

Let $\mathit{Var}(p)$ denote the set of program variables occurring in the probabilistic predicate p, $\mathit{Var}(s)$ the variables occurring in the statement s and let $\mathit{Var}_a(s)$ denote the set of program variables which are *assigned to* (i.e. subject to assignment) in s (x is assigned to in s if $x := e$ occurs in s for some e or when x is used as a variable parameter in a procedure call). We write $s \perp p$ if $\mathit{Var}_a(s) \cap \mathit{Var}(p) = \emptyset$ and $s \perp s'$ if $\mathit{Var}_a(s) \cap \mathit{Var}(s') = \emptyset$. Thus we call a program orthogonal to a predicate (or to another program) if the program does not change the variables used in the predicate (or in the other program).

The following theorem states that we can add and remove orthogonal statements without changing the validity of a Hoare triple. As we shall see in Section 4, this is precisely what is needed to establish the security of ElGamal.

Theorem 6. *If* $s' \perp q$ *and* $s' \perp s''$ *then* $\{\, p \,\}\ s\ ;\ s'\ ;\ s''\ \{\, q \,\}$ *is valid if and only if* $\{\, p \,\}\ s\ ;\ s''\ \{\, q \,\}$ *is valid.*

The notion of orthogonality $\perp$ is a practical and purely syntactically defined relation, and thus easy to check. On the other hand, $\perp$ does not have commonly

used properties of relations such as reflexiveness, transitivity and congruence properties. Therefore, care must be taken in reasoning with this relation outside of its intended purpose, that is to add or remove non-relevant program sections in a derivation, so one can transform a program into the exact required form.

4 Application: Security Analysis of ElGamal

We now apply our technique to derive semantic security for ElGamal [6].

ElGamal Let G be a group of prime order q, and let $\gamma \in G$ be a generator. (The descriptions of G and γ, including q, represent arbitrary "system parameters"). Let $Z_q^* = \{1, \ldots, q-1\}$ denote the usual multiplicative group. A key is created by choosing a number uniformly from Z_q^*, say $x \in Z_q^*$. Then x is the private key and γ^x the public key. To encrypt a message $m \in Z_q^*$, a number $y \in Z_q^*$ is chosen uniformly from Z_q^*. Then (c, k) is the ciphertext, for $c = m \cdot \gamma^{xy}$, and $k = \gamma^y$. To decrypt using the private key x, compute c/k^x, since $\frac{c}{k^x} = \frac{m \cdot \gamma^{x\,y}}{\gamma^{y\,x}} = m$.

Security Analysis The security of ElGamal cryptosystem is shown w.r.t. the Decisional Diffie-Hellman (DDH) assumption. Suppose we sample uniformly the values x, y and z. Fix ε_{ddh} small and RND large w.r.t. the system parameters. Then the DDH assumption (for G) states that no effective procedure $D(\cdot)$ (with randomness given by a uniform sample from $\{1, \ldots, RND\}$, encoding a finite tape of uniformly distributed bits) can distinguish triples of the form $\langle \gamma^x, \gamma^y, \gamma^{xy} \rangle$ from triples of the form $\langle \gamma^x, \gamma^y, \gamma^z \rangle$ with a chance better than ε_{ddh}.

In our formalism we do not precisely define the meanings of "small", "large", "better" and "effective", as they are not required in the actual proof transformations. However, one should keep in mind that these notions need to be defined properly, where e.g. "effective" means time bounded by a polynomial in the security parameter. Moreover, our fixed values (e.g., ε_{ddh}) implicitly depend on the arbitrary system parameters, so asymptotic bounds can be expressed properly (so in fact ε_{ddh} is negligible when the security parameter tends to infinity).

Semantic Security The semantic security game for ElGamal cryptosystem consists of the following four steps: (1). Setup: x is sampled from Z_q^* and r is sampled from RND. (2). Attacker chooses m_0, m_1 using inputs γ^x, r. (3). y is sampled from Z_q^*, bit b is sampled uniformly, and let $c = \gamma^{xy} \cdot m_b$. (4). Attacker chooses b' using inputs γ^x, γ^y, r, c.

Now, the attacker wins this game if it outputs b' equating b, that is the attacker can guess b with a non-negligible probability (in our case, better than $1/2 + \varepsilon_{ddh}$). A standard proof (e.g., the one given in [16]) reduces the security of this notion (i.e. that the attacker cannot win the game) to the DDH assumption described above. We now describe a similar proof within our formalism.

ElGamal Security Analysis in *pL* In our formalism, the DDH assumption ensures that for any effective procedure $D(\mathtt{v1}, \mathtt{v2}, \mathtt{v3}, \mathtt{v4}, \mathtt{v5}; \mathtt{x1})$ with inputs $\mathtt{v1}, \mathtt{v2}, \mathtt{v3}, \mathtt{v4}, \mathtt{v5}$ and output boolean $\mathtt{x1}$, the following is a valid Hoare triple.

$$\begin{array}{l}\{\mathbb{P}(\mathtt{true}) = 1\} \\ \quad \mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;\ \mathtt{r1} \leftarrow \mathit{RND};\ \mathtt{b1} \leftarrow \mathit{Bool};\ D(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{xy}}, \mathtt{r1}, \mathtt{b1}; \mathtt{out1}); \\ \quad \mathtt{z} \leftarrow Z_q^*;\ \mathtt{r2} \leftarrow \mathit{RND};\ \mathtt{b2} \leftarrow \mathit{Bool};\ D(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2}; \mathtt{out2}) \\ \{|\mathbb{P}(\mathtt{out1}) - \mathbb{P}(\mathtt{out2})| \leq \varepsilon_{ddh}\}\end{array}$$

Here, the extra provided randomness $\mathtt{b1}$ and $\mathtt{b2}$ to procedure $D(\cdot)$ are given solely to ease the exposition (as $\mathtt{r1}$ and $\mathtt{r2}$ already provide enough randomness).

ElGamal Semantic security We assume three attacker functions $A0(\mathtt{v1}, \mathtt{v4})$, $A1(\mathtt{v1}, \mathtt{v4})$ and $A2(\mathtt{v1}, \mathtt{v2}, \mathtt{v3}, \mathtt{v4})$. Functions $A0(\mathtt{v1}, \mathtt{v4})$ and $A1(\mathtt{v1}, \mathtt{v4})$ return two numbers $\mathtt{m0}$ and $\mathtt{m1}$ from Z_q^*. Similarly, function $A2(\mathtt{v1}, \mathtt{v2}, \mathtt{v3}, \mathtt{v4})$ returns a boolean. From these attacker functions we define another procedure $S(\mathtt{v1}, \mathtt{v2}, \mathtt{v3}, \mathtt{v4}, \mathtt{v5}; \mathtt{x1}) : \mathtt{B}_S$, where the body B_S is defined as follows:

$$\begin{array}{l}\mathtt{B}_S \stackrel{\triangle}{=} \mathtt{m0} := A0(\mathtt{v1}, \mathtt{v4}); \mathtt{m1} := A1(\mathtt{v1}, \mathtt{v4}); \\ \qquad \text{if } \mathtt{v5} = \mathtt{false} \text{ then } \mathtt{tmp} := \mathtt{v3} \cdot \mathtt{m0} \text{ else } \mathtt{tmp} := \mathtt{v3} \cdot \mathtt{m1} \text{ fi}; \\ \qquad \mathtt{b} := A2(\mathtt{v1}, \mathtt{v2}, \mathtt{tmp}, \mathtt{v4}); \\ \qquad \text{if } \mathtt{v5} = \mathtt{b} \text{ then } \mathtt{x1} := \mathit{true} \text{ else } \mathtt{x1} := \mathit{false} \text{ fi};\end{array}$$

Proving the semantic security of ElGamal amounts to establish:

Theorem 7. *The following is a valid probabilistic Hoare Triple:*

$$\begin{array}{l}\{\mathbb{P}(\mathtt{true}) = 1\} \\ \quad \mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;\ \mathtt{r1} \leftarrow \mathit{RND};\ \mathtt{b1} \leftarrow \mathit{Bool};\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{xy}}, \mathtt{r1}, \mathtt{b1}; \mathtt{out1}) \\ \{|\mathbb{P}(\mathtt{out1}) - 1/2| \leq \varepsilon_{ddh}\}\end{array}$$

To establish this result, we first show the following lemma.

Lemma 8. *The following is a valid Probabilistic Hoare Triple:*

$$\{R_{Z_q^{*3}, \mathit{RND}, \mathit{Bool}}(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2})\}\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2}; \mathtt{out2})\ \{\mathbb{P}(\mathtt{out2}) = 1/2\}$$

Proof. (Sketch) We use rule (3) from Section 3 on the definition of procedure $S(\cdot)$, to establish the validity of the following triple (We derive this triple formally in [5]):

$$\{R_{Z_q^{*3}, \mathit{RND}, \mathit{Bool}}(\mathtt{v1}, \mathtt{v2}, \mathtt{v3}, \mathtt{v4}, \mathtt{v5})\}\ \mathtt{B}_S\ \{\mathbb{P}(\mathtt{x1}) = 1/2\}$$

Now, to establish Theorem 7, we start by showing the validity of the following Hoare triple:

$$\begin{array}{l}\{\mathbb{P}(\mathtt{true}) = 1\} \\ \quad \mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;\ \mathtt{z} \leftarrow Z_q^*;\ \mathtt{r2} \leftarrow \mathit{RND};\ \mathtt{b2} \leftarrow \mathit{Bool}; \\ \{R_{Z_q^{*3}, \mathit{RND}, \mathit{Bool}}(\mathtt{x}, \mathtt{y}, \mathtt{z}, \mathtt{r2}, \mathtt{b2})\} \rightarrow \{R_{Z_q^{*3}, \mathit{RND}, \mathit{Bool}}(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2})\} \\ \quad S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2}; \mathtt{out2}) \\ \{\mathbb{P}(\mathtt{out2}) = 1/2\}\end{array}$$

The lower part of the triple is given by Lemma 8. For the upper part, we use Lemma 4(1) to obtain to obtain $\{R_{Z_q^{*3},RND,Bool}(\mathtt{x},\mathtt{y},\mathtt{z},\mathtt{r2},\mathtt{b2})\}$ from the random samples. The implication follows from standard properties of the group Z_q^* and the generator γ, which is a permutation of Z_q^* (In the long version [5] we derive formally a similar property). Finally, we combine the two triples using rule (Seq).

The next step consists in adding the orthogonal statements (shown boxed below) between the assignments of `y` and `z` of the above triple. Since the added statements are orthogonal (they assign to `r1,b1,out1` only, which do not occur in the above triple), by Theorem 6 we get that the following triple is valid:

$$\begin{array}{l}\{\mathbb{P}(\mathtt{true}) = 1\}\\ \quad \mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;\ \boxed{\mathtt{r1} \leftarrow RND;\ \mathtt{b1} \leftarrow Bool;\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{xy}}, \mathtt{r1}, \mathtt{b1}; \mathtt{out1});}\\ \quad \mathtt{z} \leftarrow Z_q^*;\ \mathtt{r2} \leftarrow RND;\ \mathtt{b2} \leftarrow Bool;\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2}; \mathtt{out2})\\ \{\mathbb{P}(\mathtt{out2}) = 1/2\}\end{array}$$

This is the DDH assumption when $D(\cdot)$ is instantiated by $S(\cdot)$. We use rule (And) and join the postconditions $\{\mathbb{P}(\mathtt{out2}) = 1/2\}$ and $\{|\mathbb{P}(\mathtt{out1}) - \mathbb{P}(\mathtt{out2})\}| \leq \varepsilon_{ddh}$:

$$\begin{array}{l}\{\mathbb{P}(\mathtt{true} = 1)\}\\ \quad \mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;\ \mathtt{r1} \leftarrow RND;\ \mathtt{b1} \leftarrow Bool;\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{xy}}, \mathtt{r1}, \mathtt{b1}; \mathtt{out1});\\ \quad \mathtt{z} \leftarrow Z_q^*;\ \mathtt{r2} \leftarrow RND;\ \mathtt{b2} \leftarrow Bool;\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{z}}, \mathtt{r2}, \mathtt{b2}, \mathtt{out2})\\ \{\mathbb{P}(\mathtt{out2}) = 1/2 \wedge |\mathbb{P}(\mathtt{out1}) - \mathbb{P}(\mathtt{out2})| \leq \varepsilon_{ddh}\} \rightarrow \{|\mathbb{P}(\mathtt{out1}) - 1/2| \leq \varepsilon_{ddh}\}\end{array}$$

The last application of rule (Cons) follows from replacing $\mathbb{P}(\mathtt{out2})$ with $1/2$. Finally, we remove the last line of statements thanks to orthogonality (as the assigned to variables do not occur elsewhere), and obtain the desired theorem:

$$\begin{array}{l}\{\mathbb{P}(\mathtt{true}) = 1\}\\ \quad \mathtt{x} \leftarrow Z_q^*;\ \mathtt{y} \leftarrow Z_q^*;\ \mathtt{r1} \leftarrow RND;\ \mathtt{b1} \leftarrow Bool;\ S(\gamma^{\mathtt{x}}, \gamma^{\mathtt{y}}, \gamma^{\mathtt{xy}}, \mathtt{r1}, \mathtt{b1}, \mathtt{out1});\\ \{|\mathbb{P}(\mathtt{out1}) - 1/2| \leq \varepsilon_{ddh}\}\end{array}$$

5 Conclusions and Future Work

Cryptographic proofs are complex constructions that use both cryptography and programming languages concepts. In our opinion, both communities can benefit from our approach: First, Hoare logic is well known in the programming languages community, and has been used to prove algorithm correctness for more than three decades. There are readily available computer aided verification systems that can handle Hoare logic reasoning systems (e.g., HOL [14], PVS [13], Coq [7]). Second, developing cryptographic proofs as games is well known in the cryptographic community [2, 9, 16]. Our logic allows to derive correctness proofs directly from these imperative programs, without code modifications.

Future Work There are several possible directions for future work. A short term goal is to cover more complex examples [2, 9, 16]. This would probably require to refine the notion of equivalence between Hoare triples to equivalence *up-to* ε,

to model transitions based on "bad events unlikely to happen" instead of the standard equivalence that models transitions based on pure indistinguishability.

The price to pay for rigorousity is in proof length, as the detailed proofs can quickly become lengthy. An axiomatization of the logic along with a library of ready-to-use proofs for standard constructions would help into reducing the complexity and proof length (this is a matter of ongoing work). Along the same lines, a longer term goal is to develop an implementation on a theorem prover to provide machine-checkable cryptographic proofs, following e.g. earlier work on (standard) Hoare logic formalization [13, 7, 1]). Here axioms and pre-computed proofs would also greatly increase efficiency and usability.

Acknowledgements We thank Pieter Hartel, Sandro Etalle, Jeroen Doumen, and the anonymous reviewers for helpful comments.

References

1. P. Audebaud and C. Paulin. Proofs of randomised algorithms in coq. In *MPC'06*.
2. M. Bellare and P. Rogaway. The game-playing technique, December 2004. At `http://www.cs.ucdavis.edu/~rogaway/papers/games.html`.
3. B. Blanchet. A computationally sound mechanized prover for security protocols. In *IEEE Symposium on Security and Privacy*, Oakland, California, 2006.
4. D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. In *CRYPTO'01*, pages 213–229. Springer-Verlag, 2001.
5. R. Corin and J. den Hartog. A probabilistic hoare-style logic for game-based cryptographic proofs (long version, `http://eprint.iacr.org/2005/467`). 2006.
6. T. ElGamal. A public-key cryptosystem and a signature scheme based on discrete logarithms. *IEEE Transactions on Information Theory*, IT-31:469–472, 1985.
7. J.-C. Filliâtre. Why: a multi-language multi-prover verification tool. Technical report, LRI, Université Paris Sud, 2003.
8. D. Galindo. Boneh-franklin identity based encryption revisited. In *ICALP*, pages 791–802, 2005.
9. S. Halevi. A plausible approach to computer-aided cryptographic proofs, 2005. At `http://eprint.iacr.org/2005/181/`.
10. J.I. den Hartog. *Probabilistic Extensions of Semantical Models*. PhD thesis, Vrije Universiteit Amsterdam, 2002.
11. J.I. den Hartog and E.P. de Vink. Verifying probabilistic programs using a Hoare like logic. *Int. Journal of Foundations of Computer Science*, 13(3):315–340, 2002.
12. C.A.R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12:576–580, 1969.
13. J. Hooman. Program design in PVS. In *Workshop on Tool Support for System Development and Verification*, Germany, 1997.
14. M.J.C. Gordon. Mechanizing programming logics in higher-order logic. In *Proc. of the Workshop on Hardware Verification*, pages 387–439. Springer-Verlag, 1988.
15. A. Ramanathan, J. C. Mitchell, A. Scedrov, and V. Teague. Probabilistic bisimulation and equivalence for security analysis of network protocols. In *FoSSaCS*, pages 468–483, 2004.
16. V. Shoup. Sequences of games: a tool for taming complexity in security proofs, May 2005. At `http://www.shoup.net/papers/games.pdf`.
17. S. Tarento. Machine-checked security proofs of cryptographic signature schemes. In *ESORICS*, pages 140–158, 2005.