ETH Zurich, Department of Computer Science
SS 2017

Dr. Martin Hirt
Chen-Da Liu Zhang

Cryptographic Protocols
Notes 2

Scribe: Sandro Coretti (modified by Chen-Da Liu Zhang)

> *About the notes:* These notes serve as written reference for the topics not covered by the papers that are handed out during the lecture. The material contained therein is thus a *strict* subset of what is relevant for the final exam.

These notes discuss the complexity-theoretic formalization of interactive proofs and their computational power as well as three applications of interactive proofs: for digital signature schemes, as identification protocols (see Exercise 2.2), and (very briefly) in multi-party computation, which will be treated in more detail later in this course.

2.1 Algorithms

Algorithms are commonly formalized as Turing machines, which are the basis for defining notions such as running time and space complexity; the exact formalism, however, is of no concern in this lecture. For our purposes, an algorithm A takes some input $x \in \{0,1\}^*$, performs some computation, and outputs a value $A(x)$. Running time and space complexity of an algorithm are measured as a function of the length $|x|$ of its input x. An algorithm is called *efficient* if its running time is bounded by a polynomial (and thus also its space complexity); an algorithm with an arbitrary running time is called *unbounded.* An algorithm with access to uniform random bits during its computation is called *probabilistic* or *randomized* and *deterministic* otherwise.

2.2 Decision Problems and Languages

In theory of computation, decision problems (e.g., deciding whether some graph has a Hamiltonian cycle, whether some number z is a quadratic residue modulo some other number m, whether two graphs are isomorphic) are formalized via membership in formal languages. A *language* L is simply a subset of the set of all finite bitstrings, i.e., $L \subseteq \{0,1\}^*$.

Any well-defined decision problem can be phrased as the problem of deciding whether some *word* $x \in \{0,1\}^*$ is a member of some language L (via an appropriate encoding). Conversely, every language L defines a decision problem, namely that of deciding whether or not some x is in L.

Languages for which membership can be decided efficiently are in the complexity class $\mathbf{P}$. More precisely:

Definition 2.1. *A language L is in* **P** *if there exists an efficient algorithm A such that for all $x \in \{0,1\}^*$,*

$$A(x) = 1 \iff x \in L.$$

2.3 Non-Interactive Proofs

Proofs of statements of the type "x is a member of L" are modeled as bitstrings $w \in \{0,1\}^*$. A verification algorithm V takes as input x, representing the statement that $x \in L$, as well as a proof w and outputs a bit indicating whether it considers w a valid proof. The running time of a verification algorithm is measured only in $|x|$ (and not in $|x| + |w|$).

An important class is the class **NP** of languages L that have efficiently verifiable proofs. Traditionally the class **NP** is defined as the class of languages that are accepted by non-deterministic polynomial Turing machines. However, in this notes we will consider the witness-based definition.[1]. More precisely, we say:

Definition 2.2. *A language L is in* **NP** *if there exists an efficient (deterministic) verification algorithm V such that for all $x \in \{0,1\}^*$:*

(i) Completeness: *If $x \in L$, there exists $w \in \{0,1\}^*$ such that $V(x; w) = 1$.*

(ii) Soundness: *If $x \notin L$, for every $w \in \{0,1\}^*$, $V(x; w) = 0$.*

A bitstring w with $V(x, w) = 1$ for some x is also called a *witness for $x \in L$.* The **NP**-verifier V defines a relation $R(\cdot, \cdot)$ with $R(x, w) = 1$ if and only if $V(x, w) = 1$.

2.4 Interactive Proofs

Interactive proofs are a generalization of non-interactive proofs in that the proof string w is replaced by interaction with an *unbounded* prover algorithm P. Moreover, both P and V may now be probabilistic algorithms, and V is allowed to "make mistakes." Given an input $x \in \{0,1\}^*$, $P(x)$ and $V(x)$ interact, and, at the end of their interaction, V outputs a bit indicating of whether he accepts or not.

Definition 2.3. *An interactive proof for a language L is a pair of probabilistic algorithms (P, V), where V is efficient, such that for all $x \in \{0,1\}^*$:*

(i) Completeness: *If $x \in L$, then V accepts the interaction with P with probability at least $p = 3/4$.*

(ii) Soundness: *If $x \notin L$, then V accepts the interaction with any algorithm P' with probability at most $q = 1/2$.*

Languages with efficiently verifiable interactive proofs are in the class **IP**.

Definition 2.4. *A language L is in* **IP** *if there exists an interactive proof (P, V) for L.*

As shown in Exercise 3.1, the choice of p and q is arbitrary, as long as $0 < q < p \leq 1$.[2] Moreover, P can equivalently be restricted to be deterministic, while it is essential that V is allowed to be randomized.[3]

Interactive proofs are (probably) more powerful than non-interactive ones. In fact, it was shown by Shamir in 1990 that the languages for which there exist interactive proofs are exactly

[1] Both definitions are equivalent, as shown in Exercise 2 subtask c).

[2] Note that $q > 0$ is important, while one can w.l.o.g. set $p = 1$.

[3] However, zero-knowledge cannot be achieved with deterministic provers.

those that can be recognized by algorithms that use only polynomial space (but potentially exponential running time).

Theorem 2.1. **IP** = **PSPACE**.

In this lecture, we will not prove the theorem and merely provide a sketch of the proof for **IP** $\subseteq$ **PSPACE**, which is the easier direction.

Proof (sketch for **IP** $\subseteq$ **PSPACE***).* Let $L \in$ **IP**. Then there exists an interactive proof (P, V) satisfying the properties of Definition 2.3. One can show that there exists an algorithm A that takes an input x and computes the acceptance probability of $V(x)$ in polynomial space, which obviously allows to decide whether or not $x \in L$. □

An example of a language that is in **IP** but (probably) not in **NP** is GNI.

2.5 Applications of Interactive Proofs

2.5.1 Identification Protocols

An identification protocol allows an entity Peggy to identify herself to another entity Vic, provided that Vic is in possession of an authenticated reference value that is somehow assigned to Peggy. In the context of authentication over a communication channel, the reference value is digital. It may be, e.g., be a shared secret key or a public key for which Peggy knows the corresponding secret key. The advantage of a public-key-based identification protocol is that the same public key can be used for arbitrary applications and, thus, serve as universal digital representative of Peggy.

Such a scheme can be constructed based on any public-key encryption (PKE) scheme by requiring Peggy to decrypt a randomly chosen challenge message encrypted under her public key, which proves that she knows the private key. Alternatively, one can employ a digital signature scheme and have Peggy sign the challenge message.

To implement this idea, however, one does not necessarily need a PKE scheme. Peggy can use an instance of a hard problem (e.g., some Hamiltonian graph) as her public key and identify herself by proving knowledge of a solution (the Hamiltonian cycle). If the protocol used is zero-knowledge, the security of the private key does not degrade, even after many repeated uses.

The art of designing such identification protocols lies in achieving high efficiency on the one hand and in choosing a computational problem whose instances can reasonably assumed to be hard on the other hand. Note, for example, that **NP**-hardness is no guarantee for the security of such a protocol, since this is worst-case complexity measure and an **NP**-hard problem may still be easy on average instances.

The first practical implementation of using interactive proofs as identification protocols was proposed by Fiat and Shamir and is treated in Exercise 2.2.

2.5.2 Digital Signatures

In this section, we show that one can construct digital-signature (DS) schemes from interactive proofs *of knowledge* using the so-called *Fiat-Shamir heuristic.*

Consider first the problem of transforming an interactive proof of knowledge for an **NP**-relation $R(\cdot, \cdot)$ into a non-interactive one while maintaining the zero-knowledge property. That is, Peggy wants to generate a proof that can later be verified by Vic without interaction with Peggy yet does not reveal Peggy's secret to Vic.

The problem one faces when turning an interactive proof (of the type seen in the previous sections) into a non-interactive one is that Peggy has to choose the challenges herself, and therefore the proof will not be convincing. The solution to this dilemma works as follows: For an appropriately chosen number s of rounds, Peggy proceeds as follows:

1. For $i = 1, \dots, s$, Peggy generates the first message t_i of the i^{th} round.
2. Peggy computes the s challenges $c_1, \dots, c_s$ by applying a (cryptographic) hash function h to the $t_1, \dots, t_s$ generated in step 1, i.e., she computes $(c_1, \dots, c_s) = h(t_1, \dots, t_s)$
3. Peggy generates the s answers $r_1, \dots, r_s$ to the challenges $c_1, \dots, c_s$, for which she uses the witness w.

This non-interactive proof can be used as a DS scheme as follows: To generate a key pair, Peggy chooses (in some canonical way) a random instance x of the underlying **NP**-problem along with a witness w; the public key is x, and the secret key is w. She signs a message m by executing the above procedure to prove knowledge of the secret key w but adds the m to the list of arguments of the hash function h, i.e., she computes the challenges as $(c_1, \dots, c_s) = h(t_1, \dots, t_s, m)$. A signature for m is then the tuple $(t_1, \dots, t_s, c_1, \dots, c_s, r_1, \dots, r_s)$. The signature can be verified by first checking that the challenges have been correctly computed and then that the answers $r_1, \dots, r_s$ are correct, for which the public key x is required.

The security of this scheme stems from the fact that Peggy cannot sufficiently influence the challenge bits $c_1, \dots, c_s$, and, therefore, if s is large enough,[4] she cannot prepare herself for all of them. This necessitates, however, that the hash function not possess any structure that could be exploited by Peggy. The exact properties required the hash function needs to have in order for the resulting DS scheme to be secure depend on the interactive protocol.

A property that is sufficient is the seemingly reasonable assumption that the hash function behaves as a so-called *random oracle*, which can be thought of as a (shared) random function; the corresponding idealized world is called the *random-oracle model* (ROM). The security of a great number of cryptographic schemes can be proven in the ROM. However, a (family of) hash function provably cannot satisfy this property, not even in a cryptographic sense.[5]

Thus, the security of this scheme is heuristic. From a practical perspective, however, such schemes are very attractive. Firstly, because they can be made very efficient, and, secondly, because there is a plethora of choices for the underlying hardness assumption, which makes the Fiat-Shamir heuristic an alternative to, e.g., the RSA DS system, which is based on the hardness of factoring.

2.5.3 Multi-Party Computation

Interactive proofs will also be very useful for constructing protocols for multi-party computation (MPC), where it is often necessary that some participant of the protocol prove that he correctly executed the protocol instructions. More details will be provided in the MPC part of this lecture.

[4] If a protocol with a binary challenge space (such as the GI or Fiat-Shamir protocols) is used, then $k \approx 100 \dots 200$ is needed; if, e.g., the Schnorr protocol is used, $s = 1$ is sufficient.

[5] One can show that there exists a DS scheme that is provably secure in the ROM (based on a cryptographic hardness assumption), but any possible instantiation of the random oracle with an actual hash function leads to an insecure DS scheme in the plain model.