to be witness-hiding (see Section 4.6). Hence the resulting identification scheme has constant round complexity. We remark that for identification purposes it suffices to perform Construction 4.7.9 superlogarithmically many times. Furthermore, also less repetitions are of value: when applying Construction 4.7.9 $k = O(\log n)$ times, and using the resulting protocol in Construction 4.7.8, we get a scheme (for identification) in which impersonation can occur with probability at most 2^{-k}.

Identification schemes and proofs of ability

As hinted above, a proof of knowledge of a string (i.e., the ability to output the string) is a special case of a proof of ability to do something. It turns out that identification schemes can be based also on the more general concept of proofs of ability. We avoid defining this concept, and refrain ourself to two "natural" examples of using a proof of ability as basis for identification.

It is an everyday practice to identify people by their ability to produce their signature. This practice can be carried into the digital setting. Specifically, the public record of `Alice` consists of her name and the verification key corresponding to her secret signing key in a predetermined signature scheme. The identification protocol consists of `Alice` signing a random message chosen by the verifier.

A second popular means of identification consists of identifying people by their ability to answer correctly personal questions. A digital analogue to this practice follows. To this end we use pseudorandom functions (see Section 3.6) and zero-knowledge proofs (of membership in a language). The public record of `Alice` consists of her name and a "commitment" to a randomly selected pseudorandom function (e.g., either via a string-commitment to the index of the function or via a pair consisting of a random domain element and the value of the function at this point). The identification protocol consists of `Alice` returning the value of the function at a random location chosen by the verifier, and supplying a zero-knowledge proof that the value returned indeed matches the function appearing in the public record. We remark that the digital implementation offers more security than the everyday practice. In the everyday setting the verifier is given the list of all possible question and answer pairs and is trusted not to try to impersonate as the user. Here we replaced the possession of the correct answers by a zero-knowledge proof that the answer is correct.

4.7.5 Strong Proofs of Knowledge

Definition 4.7.2 relies in a fundamental way on the notion of *expected* running-time. We thus prefer the following more stingent definition in which the knowledge extractor is required to run in *strict* polynomial-time (rather than in *expected* polynomial-time). (We also take the oppertumity to postulate – in the definition – that no-instances are accepted with negligible probability; this is done by extending the scope of the validity condition also to x's not in L_R.)

Definition 4.7.10 (System of strong proofs of knowledge): *Let R be a binary relation. We say that an interactive function V is a* strong knowledge verifier for the relation *R if the following two conditions hold.*

- Non-triviality: *As in Definition 4.7.2.*
- Strong Validity: *There exists a negligible function $\mu : \mathbb{N} \mapsto [0,1]$ and a probabilistic* (strict) polynomial-time *oracle machine K such that for every interactive function P and every $x, y, r \in \{0,1\}^*$, machine K satisfies the following condition:*

  Let $p(x)$ and $P_{x,y,r}$ be as in Definition 4.7.2. Then, if $p(x) > \mu(|x|)$ then, on input x and access to oracle $P_{x,y,r}$, with probability at least $1/2$, machine K outputs a solution $s \in R(x)$.

  The oracle machine K is called a strong knowledge extractor.

An interactive pair (P, V) so that V is a strong knowledge verifier for a relation R and P is a machine satisfying the non-triviality condition (with respect to V and R) is called a system for strong proofs of knowledge *for the relation R.*

Sequentially repeating the (zero-knowledge) proof systems for Graph Isomorphism (i.e., Construction 4.3.6), sufficiently many times yields a strong proof of knowledge of isomorphism. The key observation is that each application of the basic proof system (i.e., Construction 4.3.6), results in one of two possible situations depending on whether the verifier asks to see an isomorphism to the first or second graph. In case the prover answers correctly in both cases, we can retreive an isomorphism between the input graphs (by composing the isomorphisms provided in the two cases). In case the prover fails in both cases, the verifier will reject regradless of what the prover does from this point on. Specifically, the above discussion suggests the following construction of a strong knowledge extractor (where we refer to repeating the basic proof systems n times and set $\mu(n) = 2^{-n}$).

Strong knowledge extractor for graph isomorphism: On input (G_1, G_2) and access to the prover-strategy oracle P^*, we procced in n iterantions, starting with $i = 1$. Initially, T (the trascript so far), is empty.

1. Obtain the intermediate graph, G', from the prover strategy (i.e., $G' = P^*(T)$).
2. Extract the prover's answer to both possible verifier moves. That is, for $j = 1, 2$, let $\psi_j \leftarrow P^*(T, j)$. We say that ψ_j is *correct* if it is an isomorphism between between G_j and G'.
3. If both ψ_j's are correct then $\phi \leftarrow \psi_2^{-1}\psi_1$ is an isomorphism between G_1 and G_2. In this case we output ϕ and halt.

4. In case ψ_j is correct for a single j and $i < n$, we let $T \leftarrow T, j$, and proceed to the next iteration (i.e., $i \leftarrow i + 1$). Otherwise, halt with no output.

It can be easily verified that if the extractor halts with no output in iteration $i < n$ then the verifier (in the real interaction) accepts with probability zero. Similarly, if the extractor halts with no output in iteration n then the verifier (in the real interaction) accepts with probability 2^{-n}. Thus, whenever $p(G_1, G_2) > 2^{-n}$, the extractor succeeds in recovering an isomporphism between the two input graphs. A similar argument may be applied to some zero-knowledge proof systems for NP. In particular, consider n sequential repetitions of the following basic proof system for the *Hamiltonian Cycle* (HC) problem which is NP-complete (and thus get proof systems for any language in $\mathcal{NP}$). We consider directed graphs (and the existence of directed Hamiltonian cycles).

Construction 4.7.11 (Basic proof system for HC):

- Common Input: *a directed graph* $G = (V, E)$ *with* $n \stackrel{\text{def}}{=} |V|$.
- Auxiliary Input to Prover: *a directed Hamiltonian Cycle,* $C \subset E$*, in* G.
- Prover's first step (P1): *The prover selects a random permutation,* π*, of the vertices* V*, and commits to the entries of the adjacency matrix of the resulting permuted graph. That is, it sends an n-by-n matrix of commitments so that the* $(\pi(i), \pi(j))^{\text{th}}$ *entry is a commitment to 1 if* $(i, j) \in E$*, and is a commitment to 0 otherwise.*
- Verifier's first step (V1): *The verifier uniformly selects* $\sigma \in \{0, 1\}$ *and sends it to the prover.*
- Prover's second step (P1): *If* $\sigma = 0$ *then the prover sends* π *to the verifier along with the revealing of all commitments. Otherwise, the prover reveals to the verifier only the commitments to entries* $(\pi(i), \pi(j))$ *with* $(i, j) \in C$.
- Verifier's second step (V1): *If* $\sigma = 0$ *then the verifier checks that the revealed graph is indeed isomorphic, via* π*, to* G*. Otherwise, the verifier just checks that all revealed values are 1 and that the corresponding entries form a simple n-cycle. (Of course in both cases, the verifier checks that the revealed values do fit the commitments.) The verifier accepts if and only if the corresponding condition holds.*

We mention that the known (zero-knowledge) strong proofs of knowledge are all costy in round-complexity. Still, we have

Theorem 4.7.12 *Assuming the existence of (nonuniformly) one-way function, every NP relation has a zero-knowledge system for* strong *proofs of knowledge.*