Review: How to go beyond the black-box simulation barrier

Shiyu Zhao
Tsinghua University

Zihan Hu
Tsinghua University

Kai Su
Tsinghua University

Abstract

Former simulators mostly use V^* in black-box way. The paper [Bar01] gives a non-black-box simulator which utilizes the description of V^*. In this way, we can go beyond many limitations.

1 General Idea

In Zero-knowledge (ZK) interacitve proofs, the challenge of the simulator is how to convince the verifier without the witness. Generally, it's acknowledged that the simulator has two advantages. The first advantage is that, unlike in the actual interaction, the simulator has many attempts at answering the verifier's questions. It can just abort and retry if it fails and output the answer only when it succeeds. This is called *rewinding* technique. The second advantage is that, unlike in the true interaction, the simulator has access to the verifier's random-tape. This means that it can actually predict the next question that the verifier is going to ask.

However, the verifier could be malicious enough to embed a hash function or PRF in its algorithm. Then the output of the verifier remains uncontrollable in some sense, as if the result is still random for the simulator. Therefore, the second advantage is hard to be utilized if the verifier is a black-box strategy.

In all previously known zero-knowledge protocols, the simulator uses the verifier's strategy V^* as a black-box subroutine and thus are constricted to use only the first advantage. Rewinding does not guarantee a high probability of generating "easy questions"(questions simulator can reply successfully) within a few attempts. It is difficult to construct constant-round zero-knowledge proof under concurrent composition [Can+01], and it's also impossible to construct a constant-round zero-knowledge proof with *strict* rather than *expected* polynomial time simulators [BL04], and so on.

This paper [Bar01] aims to *to go beyond the black-box simulation barrier* by providing the simulator the code of the (possibly cheating) efficient verifier as an auxiliary input, that is what we call **non-black-box** simulator. The simulation definition of ZK says that verifiers can recreate the view without knowledge. Since the verifiers is supposed to know its own code, a non-black-box simulator is reasonable. The change of definition only occurs in the proof of security, which does not affect real world executions.

It is unfortunate that little can be done in reverse-engineering the description of the source code of the verifier. The code can be well-obfuscated. It can also contain OWFs so that even if we understand the functionality of the code, we would still not be able to make it produce "easy questions". Instead of trying to make the next question "easy", the **KEY IDEA** is to change the definition of "easy questions" to something trivial like "what the verifier would ask" in assistance of the verifier's code. This information is impossible to obtain during real executions, but transparent to the simulator since the code is known. The FLS technique [FLS99] is used to implement such modification.

See the appendix B for definitions and notations.

2 FLS-type Protocols

The idea of FLS is to define a set of trapdoor information such that if the prover possesses such trapdoor, it can convince the verifier even without a witness for $x \in L$. Additionally, a WI proof is needed so that the verifier wouldn't notice whether the prover is using a trapdoor or not.

First we introduce how to construct a zero-knowledge proof by FLS technique. It reduces the construction to two stages: a generation protocol and a WI proof system.

- **stage 1: Generation Protocol.**
  Both prover and verifier engage in a generation

protocol and output the transcript τ

- **stage 2: WI Proof.**
  Given some witness w, the prover is required to show $x \in L$ or $\tau \in \Lambda$ using a WI proof, where Λ is a public predetermined language.

Here Λ, generation protocol and WI proof need to be further specified. Λ depends on how the generation protocol is specified and WI proof is constructed through universal arguments(see appendix A for more). The generation protocol needs to satisfy the following two properties:

1. **Soundness**
   In order that the proof is meaningful, we requires that as long as the verifier follows the prescribed strategy, prover cannot cheat with trapdoor information τ rather than honestly proving $x \in L$. That is, $Pr[\tau \in \Lambda] < \epsilon(n)$, where ϵ is a negligible function, so that $\forall P^*$, if $x \notin L$, $Pr[\langle P^*, V\rangle(x) = 1] < \epsilon(n)$.
2. **Generation** (of trapdoors)
   There exists a simulator $S_{GenProt}$, for any verifier V^*, given public input and the description of V^*, it outputs (v, σ) which satisfies:
   (a) $\left\{\text{view}_{V_n^*}\left\langle P\left(x_n, y_n\right), V_n^*\left(x_n\right)\right\rangle\right\}_{n\in\mathbb{N}} \approx_C v$
   (b) $\tau \in \Lambda$ and σ is the witness for it

   Here (b) is the simulator's method to convince the verifier.

Suppose we have specified Λ, the generation protocol (called GenProt) and the WI proof (called WIProt), we need to show FLS-type protocols is zero-knowledge argument in general. Concretely, we need to show the completeness, soundness and zero-knowledge.

1. **Completeness**
   Naturally holds. If $x \in L$, then $\langle x, \tau\rangle \in L'$ by definition. Witness for $x \in L$ is also witness for $\langle x, \tau\rangle \in L'$.
2. **Soundness**
   It holds by the soundness of generation protocol. If $x \notin L$, since $Pr[\tau \in \Lambda] < \epsilon(n)$, then $Pr[x \in L \text{ or } \tau \in \Lambda] < \epsilon(n)$
3. **Zero-knowledge**

To prove that FLS-type protocols is **zero-knowledge**, we construct the following simulator:

- **stage 1: Simulated Generation Protocol**
  Given V'', $S_{GenProt}$ outputs (v, σ) just as generation protocol. v is the view and $\tau \in \Lambda$.
- **stage 2: Simulated WI Proof**
  Given x, τ, σ, V''' accepts or rejects by whether $x \in L$ or $\tau \in \Lambda$. v' is the view.
- **output:** (v, v')
- **Notation.** x is public input, V' is the description of verifier, V'' is V' with x hardwired in, V''' is V'' with view v hardwired in.

Note that although generation protocol doesn't use x as prescribed, but in case the prover or verifier is malicious to utilize x, x is hardwired into V' during stage 1 simulation. By the definition of S_{GenProt} and WI, the combined view is indistinguishable from that in a real-world execution.

3 GenProt with Uniform Verifiers

The trapdoors in the FLS technique corresponds to "easy questions". As mentioned in section 1, we set the trapdoor information to be related to the question that the verifier asks.

Notice that the entropy of uniform verifiers are limited by its random tape. Such weakness can be identified by Kolmogorov complexity. That is, the strings generated by the verifier can be described by some short Turing machine codes.

Protocol 1 (Kolmogorov GenProt)**.** *Define* $\tau \in \Lambda \iff \exists M, |M| < \frac{|r|}{2}$ *and* $r \leftarrow M()$ *in* $|r|^{\log\log|r|}$ *steps.*

	Prover		**Verifier**
1 :		$\xleftarrow{\quad r \in \{0,1\}^n \quad}$	

Proof. (Sketch)

Soundness. By the counting argument, there are at most $2^{\frac{n}{2}+1} - 1$ strings in Λ for a fixed n. For an honest verifier, $\Pr[\tau \notin \Lambda] = 1 - O\left(2^{-\frac{n}{2}}\right)$.

Generation. Use a PRG $\{0,1\}^{0.1n} \to \{0,1\}^*$ to generate the random input for the verifier. Then the verifier's output can be described using a string of length $0.1n$, the PRG, and the uniform verifier's code. The view is indistinguishable due to the PRG. □

This scheme is not extensible since the mechanism relies on the low complexity of a uniform verifier. Consider another scheme that directly uses the behavior of the verifier.

Protocol 2 (Uniform)**.** *Define* $\tau = (z, r) \in \Lambda \iff \exists \Pi, z = Com(\Pi)$*, and* $r \leftarrow \Pi(z)$ *in* $|r|^{\log\log|r|}$ *steps.*

	Prover		**Verifier**
1 :		$z = Com(\Pi)$ $\longrightarrow$	
2 :		$\longleftarrow$ $r \in \{0,1\}^n$	

Proof. (Sketch) The verifier loses if it fails to output a string different from the output of Π, the program sent by the prover.

Soundness. Verifier wins with high probability by sending random string since the program sent by the prover has a fixed output.

Generation. The prover sends $z = Com(V^*)$, where V^* is the verifier's description hardwired with a random tape. Then the verifier would have a same output as V^* since they both have the same input z. The view is indistinguishable in simulation due to the hiding property of commitment schemes. □

4 GenProt with Non-Uniform Verifiers

Protocol 2 doesn't work with non-uniform verifiers since the description size of the verifier can be a polynomial of the size of the input. Thus we can't send the verifier's description to the verifier itself.

The prover is required to make a commitment to the verifier so that the prover cannot cheat during real executions. However, full-fledged binding property of the commitment is not necessary. Hash functions can also guarantee some level of commitment, while controlling the length of the commitment.

In this section, the definition of GenProt is slightly modified such that $\tau \in \Lambda$ is allowed, but it's hard for the prover to obtain a witness. When $x \notin L$, if an honest verifier accepts with non-negligible probability δ, a malicious prover can simulate the verifier and use the knowledge extractor on the WI proof to obtain the witness for $\tau \in \Lambda$, a contradiction. So the soundness of this modification is ensured.

Protocol 3 (Non-Uniform GenProt)**.** *Define* $\tau = (h, z, r) \in \Lambda \iff \exists \Pi, z = Com(h(\Pi))$*, and* $r \leftarrow \Pi(z)$ *in* $|r|^{\log\log|r|}$ *steps.*

	Prover		**Verifier**
1 :		$h \leftarrow \mathcal{H}_n$ $\longleftarrow$	
2 :		$z = Com(h(\Pi))$ $\longrightarrow$	
3 :		$\longleftarrow$ $r \in \{0,1\}^n$	

Proof. (Sketch)

Soundness. If the prover can obtain a witness Π for non-negligible probability δ, it can generate two $r \neq r'$ randomly and obtain $\Pi \neq \Pi'$ from $(h, z, r) \neq (h, z, r')$. Then the collision $h(\Pi) = h(\Pi')$ is generated with probability $O(\delta^3)$, a contradiction to collision resistance.

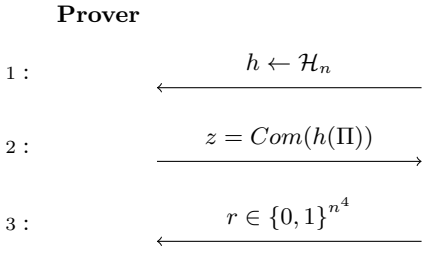
Generation. Similar to Protocol 2. □

5 GenProt with Bounded Concurrent Zero-knowledge

Roughly speaking, a protocol is bounded concurrent zero-knowledge if the verifier can not get any extra information when n statements are proved simultaneously. As verifier can jump back and forth between different sessions, its next message can depend on the response of the prover of another session. Therefore, in order to let the previous framework work, $S_{GenProt}$ is required to output the trapdoor even if several sessions interleave, which makes the previous one fail.

But wait a minute, we can put the prover's response into the trapdoor and send the generalized next message function. Formally, we give the following protocol.

Protocol 4 (Bounded Concurrent ZK GenProt)**.** *Define* $\tau = (h, z, r) \in \Lambda \iff \exists \Pi, |y| < \frac{|r|}{2}$ *such that* $z = Com(h(\Pi))$*, and* $r \leftarrow \Pi(z, y)$ *in* $|r|^{\log\log|r|}$ *steps.*



Proof. (Sketch)

Completeness. Trivial.

Soundness. Similar to Protocol 3. The fact that Π and Π' are different with high probability follows from $|y| \leq \frac{|r|}{2}$.

Bounded Concurrent ZK. Notice that the prover's responses are short and there are only n provers. Messages from other provers can fit into y. So $S_{GenProt}$ can surely output the computationally indistinguishable view along with the trapdoor even if n sessions interleave. Simulating the second stage is trivial. □

6 Summary

Using a *non-black-box* simulator, the paper presents a zero-knowledge argument system with the following properties:

1. Constant number of rounds;
2. Bounded concurrent zero-knowledge;
3. Arthur-Merlin(Public Coins);
4. Exists a simulator in **strict** poly-time.

The intuition is clearly explained by the paper.

The method embodies a novel exploitation of the non-black-box property without reverse-engineering. The description of a Turing machine can be stored and committed, as a part of the witness. It can also be sent to another party for computations under homomorphic encryptions. These are impossible for oracles. The power of non-black-box over black-box oracles can be seen as an implication of the nonexistence of VBB.

As mentioned in the last section of the paper, there remains the question that whether black-box reductions can be strengthened with the non-black-box proof. Under the context of security, non-black-box proofs are closer to the real world impossibilities.

In Fiat-Shamir paradigm, the verifier is replaced with some nice public hash function. Then everyone knows the code of the verifier. As indicated by this paper, the simulator can probably use the code to cheat, which implies the insecurity of Fiat-Shamir paradigm.

This work has some limitations. First of all, all the conclusions are based on the (seemingly strong) assumption that collision-resistant hash function against $n^{\log n}$-sized adversary exists. This limitation is overcome by [BG02]. Moreover, this work can only construct a **bounded concurrent zero-knowledge** argument system rather than a **concurrent zero-knowledge** one. Because to ensure the soundness, we have to bound the length of y. Thus y can not contain messages from a not pre-fixed polynomial number of provers.

Appendix A: Notes on Time Issues

One thing needs to be noticed. We use the description of the verifier as a witness for $\tau \in \Lambda$. But the running time of possibly malicious verifier is not prior bounded by any fixed polynomial, so the time to verify the witness is not a pre-fixed polynomial. That is, $\forall c, \Lambda \notin Ntime(n^c)$, which may lead to inefficiency. The solution is to force the time of verifying the witness to be a fixed poly in the running time of the possibly malicious verifier, and to use universal arguments, which can ensure the running time of the prover in the second stage to be a fixed poly of the time needed to verify the witness.

Why do we need the assumption that collision-resistant hash function against $n^{\log n}$-sized adversary exists? An important reason is that we can only bound Λ to be in $Ntime(T(n))$ where T is any super polynomial. Based on [Kil92] and [Mic94], under this assumption, WI universal argument system for $Ntime(n^{\log\log n})$ exists, which is a very important part of the framework.

Can this be improved? The subsequent work [BG02] analyzes the security of universal arguments in a new way, and constructs a WI universal arguments based on weaker assumption that collision-resistant hash function against poly-sized adversary exists. Finally, it shows that a slightly changed protocol can enjoy the same properties shown in [Bar01].

Appendix B: Definitions and Notations

Definition 1 (Witness indistinguishability)**.** *Let L=L(R) be some language and let (P,V) be an interactive argument for L. We say that (P,V) is witness-indistinguishable if for every polynomial-sized circuit family* $\{V_n^*\}_{n\in\mathbb{N}}$ *and every sequence* $\{(x_n, y_n, y_n')\}_{n\in\mathbb{N}}$*, where* $x_n \in \{0,1\}^n$ *and* $(x_n, y_n), (x_n, y_n') \in R$ *the following two probability ensembles are computationally indistinguishable:*

- $\left\{\text{view}_{V_n^*}\left\langle P\left(x_n, y_n\right), V_n^*\left(x_n\right)\right\rangle\right\}_{n\in\mathbb{N}}$
- $\left\{\text{view}_{V_n^*}\left\langle P\left(x_n, y_n'\right), V_n^*\left(x_n\right)\right\rangle\right\}_{n\in\mathbb{N}}$

Definition 2 (Concurrent execution)**.** *A t-times concurrent execution of a two-party protocol* (P,V) *coordinated by* V *on inputs* $\{(a_i, b_i)\}_{i=1}^t$ *is:*

1. Run t independent copies of P *with the* i^{th} *copy getting private input* b_i *and* V *getting public inputs* $a_1, \cdots, a_t$*.*

2. On each step V *outputs* (i, m)*. The* i^{th} *copy of* P *is given with the message* m *and* V *is given the prover's response.*

Definition 3 (Bounded concurrent zero-knowledge)**.** *Protocol* (P,V) *is bounded concurrent zero-knowledge for a language* $L = L(R)$ *if* $\exists$ *a p.p.t.* S *such that* $\forall$ *p.p.t.* V *and every instance* $\{(x_i, y_i)\}_{i=1}^n$ *such that* $(x_i, y_i) \in R$*,* $S(V, x_1, x_2, \cdots, x_n)$ *is computationally indistinguishable from the view of* V *in a real n-time concurrent execution of* (P,V) *on that instance.*

Definition 4 (Commitment)**.** *Denote* $Com(x, U_n)$ *as* $Com(x)$ *for simplicity. Define* $Com^{-1}(y) = x$ *if* $\exists x, r$ *such that* $Com(x,r) = y$*. Otherwise,* $Com^{-1}(y) = \perp$*.* x *is unique by the binding property.*

References

[Bar01] Boaz Barak. "How to go beyond the black-box simulation barrier". In: *Proceedings 42nd IEEE Symposium on Foundations of Computer Science*. IEEE. 2001, pp. 106–115.

[BG02] Boaz Barak and Oded Goldreich. "Universal arguments and their applications". In: *Proceedings 17th IEEE Annual Conference on Computational Complexity*. IEEE. 2002, pp. 194–203.

[BL04] Boaz Barak and Yehuda Lindell. "Strict polynomial-time in simulation and extraction". In: *SIAM Journal on Computing* 33.4 (2004), pp. 783–818.

[Can+01] Ran Canetti et al. "Black-box concurrent zero-knowledge requires\tilde {Ω}(log n) rounds". In: *Proceedings of the thirty-third annual ACM symposium on Theory of computing*. 2001, pp. 570–579.

[FLS99] Uriel Feige, Dror Lapidot, and Adi Shamir. "Multiple noninteractive zero knowledge proofs under general assumptions". In: *SIAM Journal on computing* 29.1 (1999), pp. 1–28.

[Kil92] Joe Kilian. "A note on efficient zero-knowledge proofs and arguments". In: *Proceedings of the twenty-fourth annual ACM symposium on Theory of computing*. 1992, pp. 723–732.

[Mic94] Silvio Micali. "CS proofs". In: *Proceedings 35th Annual Symposium on Foundations of Computer Science*. IEEE. 1994, pp. 436–453.