ASCrypto 2025: Recursive Proofs and Accumulation

September 29-30 of 2025

Author Emanuel Nicolás Herrador
Speaker Benedikt Bünz

TODO: Add scheme figures from slides to make explanations clear.

1 SNARKs review

We'll use SNARKs to construct our schemes here, so it's important to have this concepts in mind.

The problem to solve is: given an x, exists a w that makes the output true? The prover want to prove that he knows the solution. Here we'll working with non-interactive proofs when the proof is π and we want $|\pi| << |w|$ and $|\mathcal{V}_{\text{arg}}| << |w|$ (succinct) because otherwise we can send just the witness and it works (with completeness and knowledge-soundness). Then, verifier outputs 0 or 1 if it's true.

The properties we want are:

- Completeness: if $(x, w) \in R$ then $\mathcal{V}_{\text{arg}} \to 1$
- Soundness: if $x \notin \mathcal{L}(R)$ then with high probability $\mathcal{V}_{\text{arg}} \to 0$
- Knowledge-soundness (more general): we want to find the extractor that can access the prover and compute the witness (w). This is important in sites where the soundness is obvious.

  An example of that can be is we want to prove if we know a hash collision. Here there is a big difference between knowing existence and knowing the particular collision.

2 Recursive proofs

Consider we've two SKARK systems $(\mathcal{P}, \mathcal{V}), (\mathcal{P}', \mathcal{V}')$ (sometimes the same). Then, a recursive proof is when $\mathcal{P}'$ proves that $\mathcal{V}$ accepted the pair (x, π) (instance, proof) from $\mathcal{P}$. After that, it creates a proof π' that must be verified by $\mathcal{V}'$.

Formally, $\mathcal{P}'$ proves that it knows a proof π for a statement x in a language indexed by a verification key vk such that $\mathcal{V}$ accepts π for (x, vk) Knowledge-soundness of $(\mathcal{P}', \mathcal{V}')$ implies we can extract π. We also says $\mathcal{V}$ as the recursive circuit.

In the following, we're showing some motivations about recursive proofs use and the cons of naive solutions.

Motivation 1: Prove sequential computations Here, we've a function F that its applied t times to our instance. Then, we can break it in parts and each one has an specific witness. The definition for an intermediate instance is $x_t = F^t(x_0; w_1, \ldots, w_t)$ and we want to prove that we know this $w_1, \ldots, w_t$.

We can made a *naive solution* using SNARKs as a monolithic proof (1 proof). The instance por $\mathcal{P}$ will be x_t and then it give us a proof π. I.e., instead of dividing F computation t times, we group all the instance together and therefore in this scheme we can avoid proof-size linearity in t.

The issues of this solution is that it's not memory-efficient (because I need to store all the intermediate computation in memory to create a proof). Also, sometimes the prover is super-linear in $t \cdot |F|$. And the biggest issue is that additional steps requires reproving everything (it's bad because we don't save intermediate proofs to use it later, we should here recompute the entire history proof). Then, the proofs are linear in the entire history of the computation and we don't want it.

Motivation 2: Handing off computation It's a similar description as before but here it isn't a line and each party makes the computation apart. Here, each part wants to verify the inputs and some wants to check the entire computation. The *naive solution* is creating a proof (each party) and attach them. However it will be linear in the number of steps.

2.1 Incrementally verifiable computation (IVC)

It's a way to solve the last problems efficiently avoiding this naive solutions. In each step, when F is using evaluating value x_i then it also creates the instance x_{i+1} and the proof π_{i+1}. Finally, the last proof π_t is sent to the verifier (notice that each step is the same prover $\mathcal{P}$).

We want:

- Completeness: Given valid proof π_{i-1} for x_{i-1}, $\mathcal{P}$ generates a valid proof π_i for $x_i := F(x_{i-1}, w_i)$
- Knowledge-soundness: Given a valid proof π_i for x_i, extract witnesses $w_1, \ldots, w_t$ such that the following holds: $x_t = F^t(x_0; w_1, \ldots, w_t)$
- Efficiency: Proof-size and prover/verifier-runtime should be independent of t

Remark. The generalization of IVC is PCD (Proof Carrying Data) to permits arbitrary DAGs structures. It needs upper bound on arity.

IVC from recursive composition of SNARKs In each step we want a relation R where we say that x_{t+1} is well constructed from x_t and also π_t is verified (a verifier outputs 1). This relationship is constructed as a circuit that will be used for the ICV prover. From R is constructed our SNARK prover $\mathcal{P}$ used by our IVC module $\mathcal{P}_F$ and with our verification key Vk.

Remark. The circuit has a special case for the first module where isn't necessary a proof (because we don't have it).

The verifier IVC $\mathcal{V}_F$ gets x_1, x_t and π_t and should output the verification result. I need the first and last step of the computation.

For the properties, we've that:

- Completeness: Follows from SNARK completeness
- Soundness: Recusively extract transcript using SNARK knowledge-soundness (using the verifier parameters, we can reconstruct the instance and proofs used in steps before made)
- Efficiency: $|\pi|$ is the size of a SNARK proof for R. We want that the size of the SNARK proof doesn't grow and. Then, it's necessary sublinear verification to bound R to avoid growing up each time we construct a new proof instance.

Notice that we're using SNARKs to avoid interactivity because we can't construct our R circuits with interactivity.

We can see some applications in the follow examples.

Application 3: Property preserving SNARKs The goal here is to improve SNARK prover properties. We've a SNARK A that has a fast sequential prover (non parallel) with large memory, CRS and verifier. We want to convert it to a new SNARK A' with a fast parallel prover with constant memory, CRS and verifier-size.

The solution is to break up function F into T uniform steps F' of size $\frac{|F|}{T}$.

Application 4: SNARK composition The goal is combining SNARKs with different tradeoffs. We've SNARK A (fast prover, slow verifier, large proofs) and SNARK B (slow prover, fast verifier, small proofs, ZK) and we want a SNARK C (fast prover and verifier, small proofs, ZK).

The solution is to use SNARK B to prove correctness of SNARK A.

Many more applications There're a lot of other applications like image provenance, verifiable delay functions, succinct blockchains, ZK cluster computing, signature aggregation, and more.

Recursive proofs are widely deployed in real world (succinct, zsSync, polygon, Nexus, etc.).

2.2 Security issues for IVC

Arithmetizing $\mathcal{V}$ The problem is that as R contains $\mathcal{V}$, then $\mathcal{V}$ can't contains oracles because we need to implement $\mathcal{V}$ as a circuit.

For that, we've to have in mind the following. The *security jump* is purely heuristic (there isn't a proof of that and also it is generally not true) and it says that if we've $(\mathcal{P}^\rho, \mathcal{V}^\rho)$ secure in RO then $(\mathcal{V}, \mathcal{P})$ = *Fiat-Shamir*$(\mathcal{P}^\rho, \mathcal{V}^\rho)$ is secure in the standard CRS model. Also, there're a recent attack on GKR that relies on evaluating FS-Hash inside proofs systems but recursion relies on this ability.

So, we can't have $\mathcal{V}$ containing an oracle because it's insecure and there exists attacks for that. The model isn't broken, the real world deployment is broken for this heuristic jump.

Extraction Extraction is the way we define Knowledge-soundness in this scheme. We say that from the last proof, we can calculate all the intermediate values. Here, the idea is use SNARK extractor to create our IVC extractor. Therefore, we can use our SNARK extractor using (π_t, x_t) to extract $\pi_{t-1}, x_{t-1}, w_{t-1}$. We can use it to get proof π_k for every k using this idea.

To extract from an internal SNARK we need to simulate a prover $\tilde{P}$ for that SNARK. The idea is that $\tilde{P}$'s profs are generated by invoking the extractor for the outer SNARKs. The problem here is that each extractor can invoke each $\tilde{P}$ up to $\text{poly}(\lambda)$ times. Thus the runtime of the extractor is $\text{poly}(\lambda)$ and therefore depth *must be constant*.

It's important the extractor runtime because in exponential time we can break a lot of cryptographic primitives. So, it's interesting and important if we're able to run the extractor in small time (polynomial time).

We've to be really careful setting our security parameters to avoid security issues. One way to decrease the depth is using a decrease depth, i.e., using a tree-based IVC (binary tree) with λ arity and constant depth so we can support $\text{poly}(\lambda)$ IVC steps. This is an old solution but still there're high security loss.

Another solution, the practitioner's solution, is don't do anything. If we assume $\varepsilon_{\text{IVC}} \approx \varepsilon_{\text{SNARK}}$ so soundness error of IVC is independent of depth. In practice we do that because the security issue is just theoretical, there isn't a matching attack in practice. We can do that assuming the SNARK has a straight line (deterministi, one-shot) extractor. This is called saving grace (straightline extraction). Then, we don't get the exponential blowup (because each extractor is called once), the union bound is depth $\cdot\, \varepsilon_{\text{SNARK}}$ and recently it was improved to $\varepsilon_{\text{PCD}} \approx \varepsilon_{\text{SNARK}}$.

One problem with that is that we're only able to construct straight-line extraction in idealized models (this is our heuristic assumption). The second problem is that some SNARKs of interest don't have straightline extractors (e.g., SNARKs from non efficiently decodable codes). Therefore, straightline extraction (in ideal model) should become the norm for SNARKs (it's a recent progress of research).

Other open security problems There're another security issues as the following:

- Build IVC/PCD in standard model
- Build a model that captures Fiat-Shamir attacks but enables proving security of known SNARK/PCD constructions
- Attacks against high-depth IVC/PCD (even contrived)
- Straightline extraction in standard CRS model (or similar condition)
- Proving straightline extraction for more protocols

2.3 Efficiency

We're going to talk about practical efficiency supposing it's secure (avoiding the last security issues that we mentioned before).

IVC from succinct arguments In the construction of $\mathbb{P}$, we call for $\mathcal{P}_{\text{arg}}$ construction that $\mathcal{V}_{\text{arg}}$ is a recursive overhead because this circuit is additional to the one we want (F). The concrete issues here are that $\mathcal{P}_{\text{arg}}$ is slow and $\mathcal{V}_{\text{arg}}$ is large (because we've to construct it as a circuit). Ideally, we only want to evaluate F and not to do more much work of that.

Then, recursive overhead is a bottleneck. If we saw the way to *improve SNARK prover properties* we mentioned in last subsections, we can see that the larger is T, better are the properties obtained but worst is the verifier size obtained. So, we want to work in this recursive overhead to allow us to use a large T.

The other problem is that SNARK prover is a bottleneck. PCD prover runs SNARK prover that have large constants and strong assumptions (e.g., SNARKs in DLOG groups). Also, most efficient SNARKs have large proofs (linear-time SNARKs have MB sized proofs and leads to large recursive overheads). With that, we've a question: Are SNARKs necessary to bould IVC/PCD? We'll see another way in the following section.

3 Accumulation

We'll focus now on SNARGs.

Reductions We're using also reductions, where prover and verifier create another x' instance and w' witness from x, w, π to convert $(x, w) \in_? R$ into $(x', w') \in_? R'$. The idea is to create a way to construct that such that if $(x', w') \in_? R'$ then, the another relationship works. The idea here is to have R' as the trivial language (binary one), so the check is trivial (if x is equal to 1).

The properties are similar as SNARKS:

- Completeness: if $(x, w) \in R$ then $(x', w') \in R'$
- Soundness: if $x \notin L(R)$ then with high probability $x' \notin L(R')$. More general, we have knowledge-soundness.

Accumulation schemes Here, we've the same idea as before but now we want to reduce the proposition $(x, w) \in_? R \wedge (x_\star, w_\star) \in_? R_\star$ to $(x'_\star, w'_\star) \in_? R_\star$. This is great for us because we're able to reduce lot of properties to only one.

The accumulation scheme for R is a reduction from $R \times R_\star$ to $R_\star$ using this idea multiple times. More precisely, this is called a split accumulation (or folding) scheme. This, also, can be generalized to $R^n \times R_\star^m \to R_\star$.

Reduction of knowledge is more general than arguments, so it's weaker than an argument for R and constructing a scheme for that is, therefore, difficult. We want to solve this point, getting more interesting reductions than arguments for R. This is important because this is an easier problem to solve than constructing an argument scheme.

IVC from accumulation schemes From accumulation schemes we're able to construct IVC. Attach to the accumulation scheme the module of F, so the recursive module is now formed by F and $\mathcal{V}_{\text{acc}}$. So, we've that $(x_{i+1}; w_{i+1}) \in R \iff \mathcal{V}_{\text{acc}}(x_i; x_{\star,i}, \pi) = x_{\star,i+1} \wedge F(\text{st}_i) = \text{st}_{i+1}$.

Then, we will have a $\mathcal{P}_{\text{NARK}}$ formed by F and $\mathcal{V}_{\text{acc}}$ such that this also outputs x_{i+1}, w_{i+1} and it's $\mathbb{P}$. The verifier $\mathbb{V}$ will be formed by checking $(x_T, y_T) \in R \wedge (x_{\star,T}, y_{\star,T}) \in R_\star$

The high level idea is reduce checking this proof to a reduced one. We're proving that this is right, so we only have to check this last reduction and if it's accepted, then all the another reductions works well. Only in the very end, we've to do the check, because if we cheated or fails a reduction, then in the last check we won't have $(x_{\star,T}, w_{\star,T}) \in R_\star$.

Why accumulate? Comparing IVC from accumulation with IVC from succinct we're able to see that $\mathcal{P}_{\text{acc}} + \mathcal{P}_{\text{NARK}}$ can be faster than $\mathcal{P}_{\text{arg}}$ and also $\mathcal{V}_{\text{acc}}$ smaller than $\mathcal{V}_{\text{arg}}$. Therefore, we solved the problems we mentioned before.

Accumulation is simpler than SNARKs (we can construct it in settings an with efficiencies that don't admit SNARKs) and it suffices to build IVC/PCD.

Building accumulation The whole point of this is that construction an accumulation scheme is "easy". We will see that with examples.

In the first example, the problem is to check $a, b, c : a_i + b_i = c_i \in \mathbb{F}\ \forall i \in [n]$.

For $\mathcal{P}_{\text{NARK}}$ we will have as output $x = \text{Commit}(a, b, c)$ and $w = (a, b, c)$. The commitment can be built from DLOG and the important property is that $\text{Commit}(w) + \text{Commit}(w') = \text{Commit}(w + w')$ and also it has fixed size (it's homomorphic). Then, we want to prove $(x, w) \in R : \{x = \text{Commit}(a, b, c) \wedge a + b - c = 0\}$.

The accumulation scheme we construct will be based in the following. Lets have $(x, w), (x', w') \in R$. Then $(x, w) + Y \cdot (x', w') \in R$ (a line).

Remark. Notice that commits are a fully linear relation, so $\text{Commit}(a, b, c) + y \cdot \text{Commit}(a', b', c') = \text{Commit}(a + ya', b + yb', c + yc')$.

The scheme is now doing a challenge $\alpha \xleftarrow{\$} \mathbb{F}$ (for $\mathcal{V}_{\text{acc}}$) and getting $(x'', w'') \leftarrow (x, w) + \alpha \cdot (x', w')$. For soundness, we should notice that if they isn't in R, then the only way to have (x'', w'') correct is if it's zero, so we've a soundness of $\frac{1}{|\mathbb{F}|}$. The non-interactive form can be done with Fiat-Shamir heuristic.

Now, for the second example we want to solve a similar problem but now for multiplication, i.e., $a_i \cdot b_i = c_i$. The idea is similar but now $(x, w), (x', w') \in R \not\Rightarrow (x'', w'') \in R$. However, we've that $(a + Y \cdot a') \cdot (b + Y \cdot b') = c + Y^2 \cdot c' + Y \cdot ct$ with ct the cross term. Now the scheme is made first sending (prover) the cross term ct and then making the challenge (verifier) replying $a \xleftarrow{\$} \mathbb{F}$. So, to check it we want to check $(x'', w'') \in R_\star$ with $x'' \leftarrow x + \alpha \cdot x' || ct$ and $w'' \leftarrow w + \alpha \cdot w'$.

With that in mind, we are transforming the relationship $(x, w) \in R : \{x = \text{Commit}(a, b, c) \wedge a \cdot b - c = 0\}$ to $(x, w) \in \mathbb{R}_\star : \{x = \text{Commit}(a, b, c, ct) \wedge a \cdot b - c = ct\}$.

In general, with this ideas we can do reduction from $R \times R \to R_\star$ and $R \times R_\star \to R_\star$ is very similar. Also, multiplication and addition suffice to build accumulation for NP problems, and only cryptography needed is a

homomorphic vector commitment and Fiat-Shamir. I.e., we don't have PCPs, neither polynomial commitments, trusted setups or single commitment.