ZKPROOF WORKSHOP AT ZCON0 - 2018/06/27

SPEAKERS: Daniel Benarroch, Eran Tromer, Muthu Venkitasubramaniam, Andrew Miller, Sean Bowe, Nicola Greco, Izaak Meckler, Thibaut Schaeffer

NOTE TAKERS: Arthur Prats, Vincent Cloutier and Daniel Benarroch

Session 1- Document Overview & Feedback:

Intro - Eran

The goal is to standardize the works of different parties working with SNARKs. Need to define common methodology, definition, -- understand the trade off, to come up with a standard
This workshops are accompanied by documents. Zkproof.org to find those documents and it is an open effort. Trying to get a mechanism to get feedback, this is also an open problem.

Want to help users specify what properties of SNARKs they want or need, so that clients can ask practitioners possible things.

Libsnark comes from the academic world, but continued evolving outside academia. Contains all the fancy features, like recursive composition and many gadgets. There is a dozen frontends wrapping around libsnark. The gadget library is still competitive.

Snarky is a DSL written in OCaml. Written to be a functional replacement to libsnark, and to be more integrated to avoid mistakes. Really inspired by the functional languages.

Making those librairies and others interoperable is a big goal of this workshop. Also making the gadget reusable would be extremely useful.

Security - Muthu Venkitasubramaniam:

- Simulation paradigm arised from original work
- Every cryptog application can be modeled under a simulation agent - can even reach ideal functionality
- Provide a template for theoreticians or designers of systems to explain how zk and its properties are achieved.
- Composability of cryptographic primitives implies need to use UC framework.
- Language, terminology and notation (prover, witness, instance, etc..)

- How to write statements
- Clearly describe the properties of the scheme
- Describe the setup of the ZK scheme
- specify the construction based on combinatorial vs cryptographic parts (interesting open problem to
- What are the assumptions and proving that the security is met.

Specification:
- statements: bodeme or arithmetic circuit – you should clearly specify how you represent your statement. Should we add ram program? The consensus is no as there are too much changes
- syntax / Alg: specify algorithms are in the proof: prove alg, verify algo, setup algo (sometimes you can add trusted setup which can be included into setup) (setup: what kind of predicate, parameters, what the is going in the prover, verifier)
- properties - those are local – completeness sound and ZK
- setup: trusted (structure reference string and a random reference string) and on trusted (there is more here) what are the ramification,
- construction: combinatorial part and cryptography part – there is security implication in both side
- assumption (INISA in Europe)

(-efficiency that can potentially add here)

Security:

Want to provide a template to follow in order to explain how their zero knowledge is written in their paper (I want quantum, I do not want….) this is the motivation to start

Applications - Andrew Miller

The first draft of the document is online

Three case studies:
- Asset tracking and transfer
- Credential aggregation
- Regulation compliance of supply chain

For each of the use-cases / apps we want to have modularity of building schemes (gadgets and requirements) and focus on the security
- Desired security requirements and privacy goals
- Introduce camenisch stadler notation for gadgets + zk functionality as black box
    - None of the applications level description did not get into security parameter consideration / does not specify the program
    - The specs are good to give to an implementation team and have them implement under the hood but not worry about the black boxes
- Describe the problem that the app solves

        - Specify what is the public state, the witness, instance?
        - Describe the predicate in english and technical terms
    - Quests / Future work:
        - Formal verification for snark applications
    - Doc INCONSISTENCIES
        - Abstraction of accumulator gadget vs specific merkle tree gadget

Standardise on Camenisch-Stadler Notation

Zk { (wit) : P(stmt, wit) = 1 }

wit is the secret witness, p is a predicate, sometimes also called statement

pp <- Setup(l, p)
Pi <- Prove (pp, wit, stmt)
{0, 1} <- Verify (pp, pi, stmt)

Example: zcash-like asset

Public State: merkle tree of notes
Note: commitment {Nullifier, Pubkey, assetId}

ZK {(pubkey, pubkeyOut, merkleProof, NullifierOut, assetId, sig)}

The state transition is in the zkSNARK. It also checks that the transition was valid.

Implementation - Sean Bowe

- Middle boundary between apps and security
    - Security -> good way to test / benchmark proving systems
    - In itself
    - Apps -> ensure can use zk as black box by defining APIs
    - Two kinds of API
        - Non-universal (specific to R1CS) - setup, parameter format, prover (takes in instance and witness), verification
        - Universal API for any general language / constraint system
    - File formats such as field properties, constraints
    - Benchmarks (what kind of explanations / descriptions need to be given when making statements about the benchmark of their system. Also what other specifications) - degrees of freedom
        - Here is a constraint system check it
        - Prove a merkle tree with 128 bit security

        - Trusting the tech by ensuring some aspects (CRS, etc.)

Specifications:
- File formats for the constraint systems and metadata.
- Field properties
- Constraints (
- Discussion of the layer of metadata like
    - variable names

Benchmarks:
- security level
- criteria on how they should grade their system

constraints system or merkle tree xxxx? (choose your hash function)

Correctness and trust:
- generic list: air gaps, option for contributing….

Consensus in the group where if a cryptographic construct secures a lot of money, there is a lot more trust over time. Non consensus on if having multiple bodies check a design would help. There is nothing checking a theory against the real world.

Zcash is good use case for zero knowledge proof, because all the information comes from the blockchain. In the real world, it's much harder, because the oracle problem becomes worse. They are problems that arise from the composition of secure primitive.

Session 2- Trust and Security:

We want to focus on different topics concerning the trust of ZKP schemes and applications. These include, among others, the following list. We have generated some questions to guide the conversation.

Session moderator: Daniel Benarroch, Muthu Venkitasubramaniam

Poll the audience: why do you (not) trust Zero-Knowledge Proof based systems?
Guide the discussion to acknowledge all of the following, and try to map lay perspective and mist
- Cryptographic definitions (completeness, soundness, zero knowledge):
    - How to explain the technical definitions to a non-technical person?
    - How to convince someone that the ZKP scheme meets the definitions?

        - How to explain and convince non-technical people that the security of the scheme relies on some assumption (also how to argue about those assumptions?)
        - Example of caveats:
            - Knowledge vs Argument - the difference between “there is a witness” and “I know a witness” can be subtle, need to have further assurance than the scheme itself
            - Extractability of witness as part of the condition for catching a cheater
- Key generation / trapdoor prevention:
    - Use of trusted setup for prevention of CRS subversion
    - How do you trust that no trapdoor exists?
- Protocol caveats:
    - Defining and proving high-level domain-specific security properties
    - Common pitfall: provenance of data
        - Protocol must assure through some public verification all issues regarding data origination.
        - Must create trust that the inputs / private data are not spoofed or faked.
        - Example: proving properties of biometric data without being assured of the provenance
    - Legal context
        - How does the security definitions of the scheme delegate decisions / trust in the legal or economic context
        - Reliance of protocol on support of the legal system as a fallback mechanism (e.g., commitments as assurance of data provenance) and to recognize protocol outputs as legally binding (e.g., if the robbers hows a ZKP proof that they hold my coins, who legally owns them?)
- Trust in the provider of technology
    - How does a company prove it knows what it is doing without giving out the code? Not as simple as “use my software” since security requirements are hidden within the protocol design.
    - If we give the client the code, what can they do? Bounded rationality, limited expertise, possibility of backdoors.

MORE NOTES-

- In general the question lies in a continuous spectrum between a very technical person who would trust it by his / her own judgement by understanding the construction / security to the other end where someone who does not have the ability to understand believes it is magic and adopts it because technical people trust it
- There is a chain of trust from theoretician to implementer / provider of tech
- Outside the scheme, at protocol level
    - Technology provider
    - Legal environment / support
    - Visualization and analogies (waldo, sudoku, etc…)

    - User interface
    - Protocol UC composability or ensuring caveats (inputs etc..)
    - Bug bounties
    - More applications and adoption incentivizes the consumer / public to trust
- Inside the scheme, ZKP
    - Definitions
    - Assumptions
    - Peer review
    - Key generation

Session 3: Front-ends

Panel participants: Sean Bowe, Izaak Meckler, Thibaut Schaeffer, Eran Tromer

Moderator: Nicola Greco

Questions
- Can you share an example from your experience of an unexpected decision or change of mind you had when designing your respective front end?
- Can you share examples of feedback you have received from users writing applications in Snarky/libsnark/Bellman/ZoKrates? What is good or needs improvement?
- What level of abstraction makes sense for export/import interoperability between Frontend languages?
- What would you recommend to newcomers who want to contribute, equal reading in PL and in crypto? Or, what frontend approaches/paradigms do you think are promising but haven't yet been explored?
- There are many other frontend projects that seem somehow less well popularized, e.g. Buffet, Geppetto. I'm not sure yet how to form a productive question out of this, but i would like to acknowledge this even broader space. In a later iteration of the zkproof workshop, we plan to systematically survey front-ends (but this panel is not expected to be a survey)

MORE NOTES-
- Many different libraries - have 4 / 5 different wrong ways to implement snark systems
- Setup list of mistakes / api flaws and design based on same gadget interface
- Merge three components for witnessing variables in libsnark
- Circuit adaptability by non-determinism and conditional programming
- Forced to import libsnark into more native wrapper
- Good that there are many different implementations

- Witness generation cannot separated from constraint generation since one can screw things up
- Where do we see the implementations going? Converging or not?
    - Gadgets vs other kind of structures / terminology
    - Converge towards one API?
    - Defining usability well
    - Interoperability between many front-ends to back-ends.