Simple Soundness Proofs

Alex Kampa

Aragon ZK Research, Aragon Association, CH-6300 Zug
alex.kampa@aragon.org

October 2022 - *draft*

Abstract

We present a general method to simplify soundness proofs under certain conditions. Given an adversary $\mathcal{A}$ able to break a scheme S with non-negligible probability t, we define the concept of *trace* of a *winning configuration*, which is already implicitly used in soundness proofs. If a scheme can be constructed that (1) takes a random configuration e, being the inputs and execution environment of $\mathcal{A}$, (2) "guesses" a trace, (3) modifies e based on its guess so that the modified configuration e' is statistically indistinguishable from the original one, (4) is then able to execute $\mathcal{A}$ correctly under the condition that e' is a winning configuration and that B's guess of the trace was correct, and finally (5) that during its execution $\mathcal{A}$ is unable extract any information about B's guess, then the probability of B winning can be expressed as a simple function of t and the bit-length of the trace, namely $t/2^m$. Soundness then results if 2^m is polynomial in the security parameter.

To illustrate the concept, a concrete application of this method to a simple binary voting scheme is then described in detail.

Keyword: Proof of soundness

1 Introduction

Soundness proofs tend to be quite long and technical. In this paper, we describe a general method to significantly simplify and shorten such proofs if some specific conditions are met.

2 Simple Soundness Proofs

A common method of proving the soundness of a cryptographic scheme S is the following. We first assume that there exists an adversary $\mathcal{A}$ that can break the scheme with some non-negligible probability. We then construct a scheme B which uses $\mathcal{A}$ in a simulated environment to break a known-to-be-hard problem P, also with some non-negligible probability.

Description of $\mathcal{A}$ in its native environment

$\mathcal{A}$ is a deterministic polynomial-time algorithm. There exists a finite set of *execution configurations* $E(\lambda)$ for $\mathcal{A}$, where λ is the security parameter. Each such configuration includes inputs to $\mathcal{A}$ and completely determines the execution of $\mathcal{A}$. Given a randomly selected execution configuration $e \in E(\lambda)$, $\mathcal{A}$ breaks the scheme S with probability $t(\lambda)$, where t is a polynomial function. We then say that $\mathcal{A}$ wins. In that case, e is said to be a *winning configuration.*

The scheme B_r able to correctly execute $\mathcal{A}$

We denote by B_r (where "r" stands for "real") a scheme that simulates the real execution environment of $\mathcal{A}$. Usually, this will require the knowledge of extra information about some elements of the execution configuration. For example, the execution configuration may include a sequence of group elements, and it may be necessary to know the discrete log of these elements in order to be able to always complete the execution of $\mathcal{A}$. If G is a group with q elements and generator g, a set $\{g_i\}_{i\in[n]}$ of random elements of G can be produced by first sampling $x_i \leftarrow \mathbb{Z}_q$ and outputting $\{g^{x_i}\}$. In general, it is not difficult to generate random execution environments such that B_r has all the necessary extra information.

A modified scheme B to break P

To attempt to break P using $\mathcal{A}$, the scheme B will usually need to modify the execution environment slightly. Given the modified environment e', we say that B wins if it is able to complete the execution of $\mathcal{A}$, and $\mathcal{A}$ wins. In that case, B also breaks the problem P. In some cases, however, B will be unable to complete the execution of $\mathcal{A}$. Running $\mathcal{A}$ in this simulation is therefore not equivalent of running $\mathcal{A}$ in its normal execution environment. This fact usually complicates the soundness proof, as conditional probabilities must be introduced to deal with cases where B has to abort.

Conditions for a simple soundness proof

We now describe the conditions necessary for applying our method.

When $\mathcal{A}$ wins, this is characterised by a unique *trace* of its winning configuration e, denoted $tr(e)$, which we can think of as a very small subset of its full execution trace. The maximal bit-length m of the trace must be such that 2^m is polynomial in λ. The trace will typically be a tuple of numbers, group elements, etc.

There is a well-defined procedure by which, after generating a random configuration e and "guessing" a trace tr' by randomly sampling from $\{0,1\}^m$, B is able to modify e based on tr'. This results in a new configuration e'. The following must hold:

- the probability space of e' is the same as the probability space of e;
- $\mathcal{A}$ learns nothing about B's guess during the execution, unless B aborts - at which stage B cannot win anyway so it has no impact on the result;
- if e' is a winning configuration, and B guessed the trace $tr(e')$ correctly, then B is able to finish the execution of $\mathcal{A}$ and therefore win.

Applying the method

If the above conditions are met, the configuration e' generated by B is indistinguishable from a uniformly sampled configuration of E. Therefore, with probability t, it will be a winning configuration. The trace tr' will then be equal to $tr(e')$ with probability $1/2^m$, and this will be independent of the probability that e' is a winning configuration. If B guessed the trace correctly, it will be able to complete the execution of A and, as a result, break P, with probability of at least $t/2^m$.

3 Application to a binary voting scheme

In this section, we will apply the method described above to a simple binary voting scheme, called BatRaVot, presented in [1]. This is a scheme to allow for off-chain voting by Ethereum-based DAOs (Decentralized Autonomous Organisations), with subsequent on-chain verification. The goal is to reduce the cost of voting, which can be quite high if voting is done directly on the Ethereum blockchain. Note that ballots are public, i.e. there is no privacy.

3.1 Description of the scheme

BatRaVot is a binary voting scheme, in which voters can only choose between two options. It is a straightforward extension of the BLS (Boneh–Lynn–Shacham) [2] signature scheme.

Public parameter generation The public parameters are generated using a random $s \leftarrow \{0,1\}^\lambda$, where λ is the security parameter. They consist of two groups G and G_T of order q, a generator g of G, a bilinear map $e : G \times G \to G_T$ and a hash function $H : \{0,1\}^* \to G$ modeled as a random oracle.

Election setup For each election, a unique public election identifier $id \in \{0,1\}^*$ is chosen.

We then define $g_0 = H(id, 0) := H(id||0)$ and $g_1 = H(id, 1) := H(id||1)$.

Voters There are n voters $\{V_i\}_{i\in[n]}$, each with a pair of public/secret keys $(Pk_i, Sk_i) = (g^{s_i}, s_i)$.

Ballots The ballot of voter i for choice $v \in \{0,1\}$ is $B_i(v) = g_v^{s_i}$.

A ballot is valid if $e(B_i(v), g) = e(g_v, Pk_i)$ holds.

Proof The proof is constructed as follows. For $v \in \{0,1\}$, denote by I_v the set of indices for which a voter has submitted a valid ballot for choice b. Note that I_0 and I_1 may overlap. We then define:

$$\gamma = \prod_{i\in I_0} B_i(0) \prod_{i\in I_1} B_i(1)$$

The proof is then $\pi := (\gamma, I_0, I_1)$.

This proof is valid if $e(\gamma, g) = e(H_0, g_0)e(H_1, g_1)$ holds, where $H_b = \prod_{i\in I_b} Pk_i$.

3.2 Soundness experiment

We now describe the soundness experiment with an adversary $\mathcal{A}$ which is modeled as a deterministic, polynomial-time algorithm.

Setup The public parameters are generated using $s \leftarrow \{0,1\}^\lambda$ as before.

In addition, we generate a random set of keypairs $\{Pk_i, Sk_i\}_{i\in[n]}$.

Voting $\mathcal{A}$ receives as input $(s, PK_1, ..., PK_n)$ and can then request to receive the private keys of a subset C of voters. During the voting phase, it can query a Vote Oracle $VO(id, v, i)$ for any combination of $id \in \{0,1\}^*$, $v \in \{0,1\}$ and $i \in [n]$. This models the ability to corrupt any number of voters, in addition to these in C. $\mathcal{A}$ obviously also has access to H. However, its total number of queries to each oracle is bounded by Q, which is polynomial in λ.

$\mathcal{A}$ then outputs $(id^*, m_0^*, m_1^*, \pi^*)$, where m_0^* (resp. m_1^*) is the claimed number of votes in favour of 0 (resp. 1).

If $C \neq \varnothing$, we consider that queries $VO(id^*, v, i)$ were made for all $v \in \{0,1\}$ and $i \in C$.

Winning condition $\mathcal{A}$ wins if the proof is correct and the claimed result is for more votes than should be possible given the queries to VO.

For $b \in \{0,1\}$, denote by R_b^* the set of indices i of voters for which $\mathcal{A}$ has queried $VO(id^*, b, i)$. By our assumption above, $C \subseteq R_b^*$. Given a correct proof, the winning condition can then be stated as follows: $m_0^* > |R_0^*| \vee m_1^* > |R_1^*|$.

3.3 Soundness proof

To prove the soundness of BatRaVot, we assume that $\mathcal{A}$ has a non-negligible probability t of winning. We then construct a scheme B which simulates the execution of $\mathcal{A}$ and breaks CDH with non-negligible probability.

3.3.1 A faithful simulation of the real experiment

We start with a scheme B_r that faithfully simulates the real experiment.

Setup We start with the parameter generation as above.

In addition, B_r generates $x_k \leftarrow \mathbb{Z}_q$ for $k \in [Q]$. Note that, as a result, the set $\{g_k\} = \{g^{x_k}\}$ is a uniformly sampled set of elements in G.

Simulation of the Random Oracle The RO is queried with a pair of inputs (id, v). The RO responds to the k-th distinct query with the value g_k.

Queries for inputs that were already queried are not counted, RO simply returns $\perp$.

Simulation of the Vote Oracle The Vote Oracle is queried with inputs (id, v, j) which correspond to an election identifier id, a vote v and a voter index j.

We assume that before querying $VO(id, v, j)$, $\mathcal{A}$ has already queried $H(id, v)$, which resulted in an output g_k for some $k \in [Q]$.

VO then returns $B_j(id, v) = g_k^{s_j} = g^{x_k s_j} = Pk_j^{x_k}$

Execution of $\mathcal{A}$ After receiving input $(s, Pk_1, ..., Pk_n)$, $\mathcal{A}$ requests (and receives) the private keys of voters in C, It then queries the oracles H and VO before finally outputting:

$(id^*, m_0^*, m_1^*, \pi^*)$, with $\pi^* = (\gamma^*, I_0^*, I_1^*)$

3.3.2 The trace of a winning execution

If $\mathcal{A}$ wins, there must be at least one pair $(j, v) \in [n] \times \{0, 1\}$ such that $j \in I_v^* \setminus R_v^*$. Let (j^*, v^*) be the smallest such pair in lexicographic order. We then know these facts: $VO(id^*, v^*, j^*)$ was not queried[1], which implies $j^* \notin C$. However $H(id^*, v^*)$ must have been queried[2]. Therefore there exists k^* such that $H(id^*, v^*) = g_{k^*}$.

The tuple (j^*, k^*) is the *trace* of the winning execution of $\mathcal{A}$. When $\mathcal{A}$ does not win, the trace is not defined i.e. $\perp$.

3.3.3 Defining a winning configuration

Given a randomness $s \in \{0, 1\}^\lambda$, a set of public keys $\mathcal{P} = \{Pk_i\}_{i \in [n]}$ and a set $\mathcal{G} = \{g_k\}_{k \in [Q]}$ of RO responses, the execution of the algorithm $\mathcal{A}$ is completely determined. We have seen that when B_r knows the discrete log of all the elements of $\mathcal{P}$ and $\mathcal{G}$, it will be able to correctly execute $\mathcal{A}$. For uniformly random $(s, \mathcal{P}, \mathcal{G})$, this execution will result in $\mathcal{A}$ winning with probability t. The tuple $(s, \mathcal{P}, \mathcal{G})$ is a *winning configuration* if the execution of $\mathcal{A}$ with this configuration would result in $\mathcal{A}$ winning if B_r knew the necessary secret keys.

The key observation is the following: given a random $(s, \mathcal{P}, \mathcal{G})$ for which some or all of the discrete logs of $\mathcal{P}$ and $\mathcal{G}$ are not known, the probability of it being a winning configuration remains the same - even though with limited computational resources we may not be able to determine whether it is a winning configuration or not. In other words, we apply the principle that the probability of winning does not depend on complexity or runtime. And if $(s, \mathcal{P}, \mathcal{G})$ is a winning configuration, then there exists a definite trace that is not $\perp$ and that corresponds to the execution of $\mathcal{A}$ with that configuration - even though we may not be able to compute that trace.

Note that if we had access to unbounded computational power, we could of course execute $\mathcal{A}$ and determine its trace for any configuration.

3.3.4 A slightly modified experiment

We now construct B by slightly modifying B_r. We are given two random elements $u \leftarrow G$ and $h \leftarrow G$ whose discrete logs we do not know.

[1]by the definition of the winning condition

[2]except with negligible probability

Modified Setup We add the following at the end of the setup phase:

for $j' \leftarrow [n]$, we replace $Pk_{j'}$ with u

for $k' \leftarrow [Q]$, we replace $g_{k'}$ with h

What this means is that we start with configuration $(s, \mathcal{P}, \mathcal{G})$, are then given random elements u and h, randomly sample j' and k' and end up with $(j', k', s, \mathcal{P}', \mathcal{G}')$. Although this looks like a different sample space, in fact when j' and k' are fixed, then $(s, \mathcal{P}', \mathcal{G}')$ has the same sample space and uniform distribution as $(s, \mathcal{P}, \mathcal{G})$. It therefore corresponds to the same probability space[3]. See also figure 1.

$$\begin{tikzcd}[column sep=huge]
\underbrace{(s,\mathcal{P},\mathcal{G})}_{\substack{\{0,1\}^\lambda \times G^n \times G^Q \\ \text{uniform distribution}}} \arrow[r, "{u,h,j',k'}"] & (j',k',\underbrace{s,\mathcal{P}',\mathcal{G}'}_{\substack{\{0,1\}^\lambda \times G^n \times G^Q \\ \text{uniform given fixed } j',\ k'}}) \\
B_r & B
\end{tikzcd}$$

Figure 1: From one distribution to another

The probability that this new configuration $(s, \mathcal{P}', \mathcal{G}')$ is a winning configuration has therefore not changed, i.e. it is still t. The only difference with B_r is that B does not know the discrete log of 2 elements of the configuration.

Assume that this modified configuration is a winning configuration and that the trace of $\mathcal{A}$ for that configuration is (j^*, k^*). This trace corresponds to an election identifier id^* and a choice v^*. The probability that (j', k') is equal to (j^*, k^*) is $1/nQ$. We shall now prove that in this particular case, B will always be able to complete the execution of $\mathcal{A}$.

C As B knows all the secret keys of voters $[n] \setminus \{j*\}$, and $j^* \notin C$, B is able to provide all the secret keys of voters in C to $\mathcal{A}$.

RO B can obviously respond to all queries to RO. In particular, it returns h for the k^*-th query, which is $H(id^*, v^*)$.

VO That leaves us with queries to VO.

When $(id, v) \neq (id^*, v^*)$, $H(id, v)$ must have been queried before so there exists some x_k known to B such that $H(id, v) = g^{x_k}$. B also knows the public keys of all the voters and returns $VO(id, v, j) = Pk_j^{x_k}$.

For (id^*, v^*) and $j \neq j^*$, $H(id, v)$ must have been queried before (to which B responded with h) and B knows the secret key of voter j and returns $VO(id^*, v^*, j) = h^{s_j}$.

The only remaining case is $VO(id^*, v^*, j^*)$, to which B is unable to provide a correct response. But given its trace, $\mathcal{A}$ never calls VO with these parameters.

[3]Although this seems rather obvious, for the sake of completeness a proof is provided in the Appendix

B can therefore complete the execution of $\mathcal{A}$, which yields a proof that contains at least one forged ballot. We define $J_b^* = I_b^* \setminus \{j^*\}$ for $b \in \{0,1\}$ and:

$$\gamma' = \prod_{i \in J_0^*} B_i(0) \prod_{i \in J_1^*} B_i(1)$$

It is then easy to see that $\gamma/\gamma' = h^\alpha$, where α is such that $u = g^\alpha$. In other words, given a pair $(u = g^\alpha, h)$ with unknown α, B was able to produce h^α. This is equivalent to breaking CDH with non-negligible probability of at least t/nQ. As this is considered infeasible, we conclude that our assumption that $\mathcal{A}$ wins with non-negligible probability is incorrect.

3.3.5 Behaviour of B in the general case

We have seen in 3.3.4 that B can complete the execution of $\mathcal{A}$ under the following conditions: the configuration is a winning configuration, and $(j', k') = (j^*, k^*)$, where (j^*, k^*) is the trace of that configuration. Let's look at what happens when these conditions are not met - even though strictly speaking this is not necessary for the soundness proof.

First of all there will be cases where B will simply not be able to complete the execution of $\mathcal{A}$. This may already happen at the very first step: if $j' \in C$, then B will not be able to provide the required private keys. B may also not be able to answer some specific query to VO. In all these cases, B can simply abort.

Note that even when B can complete the execution of $\mathcal{A}$, it will not be able to win if $\mathcal{A}$ does not win.

Finally, when we do have a winning configuration, there is still a theoretical possibility of B winning even if $(j', k') \neq (j^*, k^*)$. This could happen if $\mathcal{A}$ has a kind of "secondary trace" (j^{**}, k^{**}) corresponding to another forged ballot. In that case, if $(j', k') = (j^{**}, k^{**})$, then B will be able to win also.

3.3.6 Another look at the winning executions of B

In 3.3.4 we described the winning execution of $\mathcal{A}$ by B with reference to j^*, k^*, v^* and id^*. Of course, B does not know about these numbers during the execution of $\mathcal{A}$, it only finds out about them at the end. However, remember that this is under the assumption that $(j', k') = (j^*, k^*)$.

For additional clarity, we now restate this using only data that B knows.

C B knows all the secret keys of voters $[n] \setminus \{j'\}$. However, by our assumption $j' = j^*$ (which B does not know at that time yet), and $j^* \notin C$. Therefore B is able to provide all the secret keys of voters in C to $\mathcal{A}$.

RO B can obviously respond to all queries to RO. In particular, if there is a k'-th query, it returns h, which corresponds to the query for some $H(id', v')$.

But in that case, because $k' = k^*$ by our assumption, we will also have $(id', v') = (id^*, v^*)$. Again, B will not know this until the end of the execution.

VO That leaves us with queries to VO with input (id, v, j).

When $(id, v) \neq (id', v')$, $H(id, v)$ must have been queried before so there exists some x_k known to B such that $H(id, v) = g^{x_k}$. B also knows the public keys of all the voters and returns $VO(id, v, j) = Pk_j^{x_k}$.

When $(id, v) = (id', v')$ and $j \neq j'$, $H(id', v')$ was queried before, to which B responded with h. As B knows the secret key of voter j (it knows all the secret keys except those of voter j'), it is able to retrun $VO(id', v', j) = h^{s_j}$.

The only remaining case is $(id, v, j) = (id', v', j')$, to which B is unable to provide a correct response. But given that $(id', v', j') = (id^*, v^*, j^*)$, $\mathcal{A}$ never calls VO with these parameters.

4 Conclusion

We have presented a general approach to simplify soundness proofs under certain assumptions. We have then presented, in full detail, a concrete application of this method.

I would like to thank Vincenzo Iovino, Matan Prasma and Razvan Rosie for providing useful comments at various stages of this paper.

References

[1] Iovino et al. Succinct arguments of voting: Voting for decentralized autonomous organisations with low gas consumption. *unpublished manuscript*, 2022.

[2] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In *Advances in Cryptology – ASIACRYPT '01, LNCS*, pages 514–532. Springer, 2001.

Appendix: Randomly replacing values in vectors

From one uniformly random configuration to another

Let G be a finite set and n a positive integer. Let $H = G^n$ and $E = [n] \times G \times H$. The uniform distributions on these sets are denoted Pr_H and Pr_E. Elements of H are denoted $\vec{h}$ while elements of E are denoted $(k, u, \vec{g})$. Elements of a vector $\vec{v}$ are denoted $v(i)$. Given a vector $\vec{v}$, we denote by $r(\vec{v}, k, u)$ the vector obtained from $\vec{v}$ by replacing its k-th element by u:

$$r(\vec{v}, k, u)(i) = \begin{cases} v(i) & \text{if } i \neq k \\ u & \text{if } i = k \end{cases}$$

The set of all elements in H which are equal to a vector $\vec{h}$ except at some fixed index $k \in [n]$ is denoted $H(\vec{h}, k)$:

$$H(\vec{h}, k) = \{\vec{v} \in H \mid \forall i \in [n] \setminus \{k\},\ v(i) = h(i)\}$$

Note that $\forall(\vec{h}, k), |H(\vec{h}, k)| = |G|$. We further define the function ρ as follows:

$$\begin{aligned} \rho : E &\longrightarrow H \\ (k, u, \vec{g}) &\longrightarrow \vec{h} = r(\vec{v}, k, u) \end{aligned}$$

With fixed $\vec{h} \in H$ and $\kappa \in [n]$, we have:

$$Pr_E(\rho = \vec{h} \mid k = \kappa) = \frac{|H(\vec{h}, \kappa)|}{|E|} = \frac{|G|}{n|G|^{n+1}} = \frac{1}{n|G|^n}$$

As a result:

$$Pr_E(\rho = \vec{h}) = \sum_{\kappa \in [n]} Pr_E(\rho = \vec{h} \mid k = \kappa) = n\frac{1}{n|G|^n} = \frac{1}{|G|^n} = Pr_H(\vec{h})$$

Therefore the random variable ρ is uniform on H. This is not at all surprising: if we take a random vector in G^n, then select a random $k \in [n]$ and a random $u \in G$ and replace the k-th value of the vector with u, we naturally expect the result to also be random.

Extending the result

We can modify the definitions of E as follows: instead of $E = [n] \times G \times G^n$, we define $E = R \times G \times G^n$ where $R \subseteq [n]$. It is clear that we will obtain the same result.

Another observation is that if we have for example $E = [n] \times G \times G^n \times H^m \times ...$, we can extend f and f_κ in an obvious way, resulting in a uniform distribution on $G^n \times H^m \times ....$

Finally, it is also clear that the process can be repeated several times. For example, if our initial sample space is $G^n \times H^m$, we can randomly replace one or more values in G^n, then randomly replace one or more values in H^m, and the resulting distribution will remain uniformly random.

This last argument is what allows us to claim, in 3.3.4, that $(s, \mathcal{P}, \mathcal{G})$ and $(s, \mathcal{P}', \mathcal{G}')$ correspond to the same probability space: both obviously have the same sample space, and the same uniform distribution.

$\square$