1 Simulation-extractible protocol for small tags

The construction of the family of "small tags" arguments (based on [1]) will be based on techniques similar to Barak’s non-blackbox ZK construction. We recall Barak’s protocol:

\[
\begin{array}{c}
P \quad V \\
\hline
h \\
c = Com(h, s) \\
r \leftarrow \{0, 1\}^k \\
\hline
\end{array}
\]

Where UARG is a WI universal argument in which P proves to V that either \( x \in L \) or \( \exists (\Pi, s) \) s.t \( c = Com(h(\Pi), s) \land \Pi(c) = r \).

We will use a modified version of this protocol where the length of \( r \) is \( l(k) \), and during the UARG \( P \) proves that \( x \in L \) or there exists \( (\Pi, y, s) \) such that the following conditions all hold:

- \( c = Com(h(\Pi), s) \)
- \( \Pi(y) = r \)
- \( |y| \leq |r| - k \)

Where \( P \) has the freedom to choose \( y \) in order to try and match \( r \). This is still sound, because of the length difference between \( y \) and \( r \) - since \( P \) committed to \( C \) before it got \( r \), it has no more than probability \( 2^{-k} \) to "hit" \( r \) by choosing \( y \). It is also zero-knowledge if we assume \( l(k) \geq 3k \) and the commitment scheme outputs a commitment of size \( \leq 2k \). The simulator will use \( c \) as \( y \) and the length will be suitable to convince \( V \). Also, when these 3 statements hold, we say that \( ((h, c, r), (\Pi, y, s)) \in R_{sim} \).

We are now ready to describe the protocol for small tags of length \( \log n + 1 \). Fix a length \( l(n) \geq 3n \), and a tag \( \in [2n] \):
Protocol $< P_{tag}, V_{tag} >$

**Common Input:** An instance $x \in \{0,1\}^n$

**Stage 0 (Set up):**

$V \rightarrow P: \text{send } h \leftarrow H_n$

**Stage 1 (Slot 1):**

$P \rightarrow V: \text{send } c_1 = \text{Com}(0^n)$

$V \rightarrow P: \text{send } r_1 \leftarrow \{0,1\}^{tag \cdot l(n)}$

**Stage 1 (Slot 2):**

$P \rightarrow V: \text{send } c_2 = \text{Com}(0^n)$

$V \rightarrow P: \text{send } r_2 \leftarrow \{0,1\}^{(2n+1-tag)\cdot l(n)}$

**Stage 2 (UARG):**

$P$ proves to $V$ that one of the following statements is true:

1. $\exists w \in \{0,1\}^{\text{poly}(|x|)} \text{ s.t. } (x,w) \in R_L$
2. $\exists (\Pi, y, s) \text{ s.t. } ((h,c_1,r_1), (\Pi, y, s)) \in R_{sim}$
3. $\exists (\Pi, y, s) \text{ s.t. } ((h,c_2,r_2), (\Pi, y, s)) \in R_{sim}$

Note that $< P_{tag}, V_{tag} >$ is a proof of knowledge - if any prover $P^*$ convinces the honest verifier that $x \in L$, we can extract a witness $w$ in expected polynomial time. This will be used later to produce the EXT algorithm.

To prove simulation-extractability we need to construct $(\text{SIM}, \text{EXT})$. To that end, we construct a simulator $S$ that is able to generate the messages on the left hand interaction of $A$ (the ones from $P_{tag}$) when the right hand messages are coming from an "external" verifier ($S$ does not have the code of that verifier). We will then construct SIM by running internally the honest verifier $V_{\sim tag}$ and forward its messages to $S$. We will construct EXT by using $S$ to construct a stand-alone prover $P_{\sim tag}$ (Emulate $A$ while using $S$ to generate the left interaction and forwarding the right messages to an external honest verifier). We can then extract a witness using the proof of knowledge property.

The main problem is constructing $S$ can be summed up in the scheduling presented in the diagram below. The core of the problem is that $S$, trying to simulate the left interaction, has to commit to the code of the "verifier" $A$, but the messages $A$ sends on the left (its output in this context - $r_i$) are not dependent only on $c_i$, but also on the "external" messages in the right interaction. The simulation therefore cannot get away by setting $\Pi = A$ and $y = c_i$, because in the scenario in the diagram below the $r_i$ are also dependent on $\bar{r}_i$, and the simulator will not produce correct views in the universal argument stage.
Problematic scenario in the simulator $S$

\[ P_{\text{tag}} \quad \begin{array}{c} \text{AV}_{\bar{\text{tag}}} \\ \hline \end{array} \]

\[ \begin{array}{c}
\bar{h} \\
h \\
c_1 \\
r_1 \\
c_2 \\
r_2 \\
\end{array} \quad \begin{array}{c}
\bar{\bar{c}}_1 \\
\bar{c}_1 \\
r_1 \\
\bar{\bar{c}}_2 \\
\bar{c}_2 \\
r_2 \\
\end{array} \]

We can get around this technical problem by noting that for at least one of the slots, we will have $|c_i| + |\bar{r}_i| \leq |r_i| - n$. This allows us to set $y = (c_i, \bar{r}_i)$ (by the UARG stage we have it), which will give the correct answer (because $r_i = A(c_i, \bar{r}_i)$), and still respect the bound in $R_{\text{sim}}$, namely $|y| \leq r - n$.

This works because of how the lengths of the $r_i$'s depend on the tags. If $\text{tag} \neq \bar{\text{tag}}$, for at least one of the $i$'s we have $|\bar{r}_i| \leq |r_i| - l(n)$. And using the facts that $l(n) \geq 3n$ and $|\text{Com}(\alpha, s)| \leq 2n$ for $\alpha$ of size $n$, the required inequality follows.

2 From tags in $[2n]$ to tags in $\{0, 1\}^n$

Suppose we have a family of protocols $\{< P_{\text{tag}}, V_{\text{tag}} >\}_{\text{tag} \in [2n]}$ that are simulation-extractable, we can use them to construct a family of protocols $\{< P_{\text{TAG}}, V_{\text{TAG}} >\}_{\text{TAG} \in \{0, 1\}^n}$ that are simulation-extractable. The idea is to take the $\text{TAG} \in \{0, 1\}^n = (\text{TAG}_1, \ldots, \text{TAG}_n)$, and to generate run the protocol $n$ times in parallel with the small tags generated from the big tag:
Protocol \(< P_{TAG}, V_{TAG} >\)

**Common Input**: An instance \(x \in \{0,1\}^n\)

The protocol:

- for \(i \in \{1, \ldots, n\}\) (in parallel):
  1. Set \(tag_i = (i, TAG_i)\)
  2. Run \(< P_{tag_i}, V_{tag_i} >\) with common input \(x\) and length \(l(n)\)

Accept if and only if all \(n\) executions accept.

Note that this is a constant-round IP (since every \(< P_{tag}, V_{tag} >\) is, and we run them in parallel), and that if \(TAG \neq TAG\), for at least one \(i \in [n]\) we have that \(tag_i \neq \tilde{tag}_i\), which assures us of soundness. The problem is being able to simulate the \(n\) interactions on the left - the simulator cannot handle messages forwarded from an external \(V_{TAG}\), because they are too long, and it is not clear how to construct the stand-alone prover \(P^*_{TAG}\) (for the EXT procedure). The way around it is to construct a stand-alone prover for a single \(< P_{\tilde{tag}}, V_{\tilde{tag}} >\). We first consider a "many to one" simulator, which for a \(TAG = (tag_1, \ldots, tag_n)\) generates views for the parallel left interactions \(< P_{tag_1}, V_{tag_1} >, \ldots, < P_{tag_n}, V_{tag_n} >\) on the common input \(x \in \{0,1\}^n\), and the single right interaction \(< P_{\tilde{tag}}, V_{\tilde{tag}} >\) on common input \(\tilde{x} \in \{0,1\}^n\), where \(tag\) and \(\tilde{x}\) are chosen by \(A\).

\(S\) incorporates \(A\) as a sub-routine, and handles the right interaction by having \(A\) communicate with an "external" honest verifier \(V_{\tilde{tag}}\). On the left, messages are are handled by \(n\) sub-simulators \(S_1, \ldots, S_n\), each responsible for generating the messages of the respective sub-protocol. Ignoring the setup and the universal argument steps for simplicity, the interaction (with one specific scheduling) looks like this:

```
Many-to-one simulation

\begin{array}{ccc}
P & & A \\
& c_1^1 & \cdots & c_1^n & \rightarrow \\
& r_1^1 & \cdots & r_1^n & \\
& c_2^1 & \cdots & c_2^n & \leftarrow \\
& r_2^1 & \cdots & r_2^n & \\
\end{array}
```

\(\tilde{c}_1\) and \(\tilde{r}_1\) are both, respectively, within slot \(i\) of \(< P_{\tilde{TAG}}, V_{\tilde{TAG}} >\). If we have a simpler scheduling, for example, such that none of the right side
messages are within the first slot, the output of $A$ will depend only on the $c_i$’s (but on all of them together), and $S_j$ can work as follows. Let $A_j$ be an algorithm that is identical to $A$, but only outputs $r^j_1$. We let $\Pi_1 = A_j(x, \cdot)$, and the simulator $S_j$ commits to $c_1 = \text{Com}(h(\Pi_1); s)$. During the UARG stage, it will use $(\Pi_1, (c^1_1, \ldots, c^n_1), s_1)$ as a witness for $(h^j, c^j_1, r^j_1)$ (and it can commit to anything in the second slot). The case where the right side interaction is inside slot 2 is similar.

For the more complicated scheduling as in the diagram, where don’t have a ”free slot”, we again exploit the length difference trick as following. Let $\Pi_1 = A_j(x, \cdot), \Pi_2 = A_j(x, c^1_1, \ldots, c^n_1, \bar{r}^1, \cdot)$, and have $S_j$ commit to $c_1 = \text{Com}(h(\Pi_1); s)$ and $c_2 = \text{Com}(h(\Pi_2); s)$. Now, as before, send witnesses to the suitable slot as follows:

- If $tag_j > \bar{tag}$, we can set $(\Pi_1, (c^1_1, \ldots, c^n_1, \bar{r}^1), s_1)$ as witness to $(h^j, c^j_1, r^j_1)$.
- If $tag_j < \bar{tag}$, we set $(\Pi_2, (c^1_2, \ldots, c^n_2, \bar{r}^2), s_2)$ as witness for $(h^j, c^j_2, r^j_2)$.

References