Libsnark Tutorial: Basic Gadgets

Chan Nam Ngo
channam.ngo@unitn.it
University of Trento, Trento, Italy

November 16, 2018

1 Common Scenario

Let us consider the following scenario where Alice wants to prove to Bob that she is able to buy a certain amount v of his goods at price p but she does not want Bob to find out how much cash available that she has.

1. Alice commits to her available cash c_A, which gives her a commitment $C_A = \mathsf{com}(c_A; r_A)$ where r_A is the randomness of the commitments.
2. Alice then only needs to prove in zk that the following conditions are satisfied.
   (a) $C_A = \mathsf{com}(c_A; r_A)$
   (b) $c_A \geq v * p$

Let us move on and make a transaction of the deal as well.

1. Suppose Bob has previously committed as well his cash c_B which gives him $C_B = \mathsf{com}(c_B; r_B)$ where he keeps also r_B.
2. Alice then updates $c'_A = c_A - v * p$ while Bob updates $c'_B = c_B + v * p$.
3. Both of them then commit to C'_A and C'_B respectively and prove in zk that the following conditions are satisfied respectively.
   (a) $C'_A = \mathsf{com}(c'_A; r'_A)$
   (b) $c'_A = c_A - v * p$

   and

   (a) $C'_B = \mathsf{com}(c'_B; r'_B)$
   (b) $c'_B = c_B + v * p$

What about the goods amount? Simple, they can do the same for their respective amount v_A and v_B.

2 Dual Variable Gadget

Suppose that the commitment scheme is a SHA-256 hash function H, technically we need to do the following, e.g. for c_A:

1. Decompose $c_A, \ldots$ into a bit vector of 256 bits $\{c_{A,i}\}_{i=1}^{256}$, etc.
2. Pick a random bit vector $\{r_{A,i}\}_{i=1}^{256}$ of 256 bits
3. Produce a commitment as a 256 bits hash $\{C_{A,i}\}_{i=1}^{256} = \mathsf{H}(\{c_{A,i}\}_{i=1}^{256}; \{r_{A,i}\}_{i=1}^{256})$

The required conditions for c_A, as shown above is $c_A \geq v * p$ where v and p are numbers, not bit vectors. Hence it requires an additional condition where we shows c_A is consistent with $\{c_{A,i}\}_{i=1}^{256}$.

The dual_variable_gadget is useful in this case.

```
template<typename FieldT>
class dual_variable_gadget : public gadget<FieldT> {
private:
std::shared_ptr<packing_gadget<FieldT> > consistency_check;
public:
pb_variable<FieldT> packed; // value c_A
pb_variable_array<FieldT> bits; // bit vector {c_A,i}
void generate_r1cs_witness_from_packed(); // c_A to {c_A,i}
void generate_r1cs_witness_from_bits(); // {c_A,i} to c_A
};
```

3 Comparison Gadget

So far we have declared c_A as a variable, $\{c_{A,i}\}_{i=1}^{256}$ as a variable array and used dual_variable_gadget to guaranteed the consistency. We can move on and check $c_A \geq v * p$. For this purpose we will need additional variables.

As we can only describe constraints using R1CS, we have to covert all the conditions into the format $a * b = c$. As an example, we can declare x, v and p as variables and add the constraint $v * p = x$.

The rest is to compare c_A and x where we can make use of the comparison_gadget.

```
template<typename FieldT>
class comparison_gadget : public gadget<FieldT> {
const pb_linear_combination<FieldT> A;
const pb_linear_combination<FieldT> B;
const pb_variable<FieldT> less; // A < B
const pb_variable<FieldT> less_or_eq; // A <= B

void generate_r1cs_constraints();
void generate_r1cs_witness();
};
```

The example usage is as follows.

```
protoboard<FieldT> pb;

pb_variable<FieldT> A, B, less, less_or_eq;
A.allocate(pb, "A");
B.allocate(pb, "B");
less.allocate(pb, "less");
less_or_eq.allocate(pb, "less_or_eq");

comparison_gadget<FieldT> cmp(pb, n, A, B, less, less_or_eq,
"cmp");
cmp.generate_r1cs_constraints();
```

4 Multi-Packing Gadget

As we are working with bit vectors of 256 bits. It is useful to compress them into Field elements to reduce the size of the statement.

```
// Connects hasher_output with circuit output
// (this->commitment)
this->commitment_packer
= make_shared<multipacking_gadget<FieldT>>(
pb, hasher_output.bits, this->commitment, FieldT::capacity(),
FMT(this->annotation_prefix, " commitment_packer"));
```

5 Other useful gadgets

All of the basic gadgets can be found in gadgetlib1's *basic_gadgets* file.

```
disjunction_gadget // OR
conjunction_gadget // AND
inner_product_gadget
```

6 What to do now?

The following examples are provided:

1. Comparison of a value and a constant
2. Comparison of two values
3. Range check for a value

Follow the example gadgets and try to implement the following scenario.

1. Alice commits to c_A, her available cash, and v_A, her holding goods.
2. Alice proves in zk that her total value $c_A + v_A * p$ is above a threshold t, i.e. $c_A + v_A * p \geq t$ where p and t are public values.

Hints:

1. There are two commitments C_A and V_A, so two variables
2. Two variable arrays are required to bind C_A and V_A to the hashers (two hashers and two dual variable gadgets required)
3. Two variables are required for the public values p and t
4. One variable X is required to hold the value for $X = v_A * p$ (1 constraint)
5. One variable Y is required to hold the value for $c_A + v_A * p = c_A + X$ (1 constraint in the form $Y = 1 * (c_A + X)$)
6. The comparison gadget then can be used with Y and t where $B = Y$ and $A = t$ because we want to check $Y \geq t$ which means $t \leq Y$.