A Boolean Algebra Over Trapdoors

2023-06-17

Abstract

This paper introduces a Boolean algebra framework over trapdoors, establishing a novel approach to cryptographic operations within a Boolean algebraic structure. The core of the framework is the Boolean algebra $A := (\mathcal{P}(X^*), \wedge, \vee, \neg, \emptyset, X^*)$, with $\mathcal{P}$ representing the powerset and X^* the free semigroup on an alphabet X. A key feature of this study is the homomorphism $F : A \mapsto B$ from A to a Boolean algebra $B := (\{0,1\}^n, \&, |, \ , 0^n, 1^n)$ of n-bit strings, achieved through a cryptographic hash function. This homomorphism introduces a secret s into its operation, embedding security within the algebraic structure and rendering F one-way. Our exploration highlights the cryptographic utility of this framework, especially in terms of collision probability and resistance to reverse engineering, offering a foundational basis for secure cryptographic operations leveraging Boolean algebra.

Contents

Consider the Boolean algebra

$$A := (\mathcal{P}(X^*), \wedge, \vee, \neg, \emptyset, X^*)$$

where $\mathcal{P}$ is the powerset, X is the alphabet, and X^* is the free semigroup on X which is closed under concatenation,

$$\# : X^* \mapsto X^* \mapsto X^*.$$

For example, if

$$X = \{a, b\}$$

then

$$X^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, ...\}$$

and $\mathcal{P}(X^*)$ is the power set of X^*,

$$\mathcal{P}(X^*) = \emptyset, \epsilon, \{a\}, \{b\}, \{aa\}, \{a, aa\}, \{a, bb\}, ...$$

Consider the Boolean algebra

$$B := (\{0,1\}^n, \&, |, \ , 0^n, 1^n)$$

and suppose we have a homomorphism

$$F : A \mapsto B$$

defined in the following way. First, we have a cryptographic hash function

$$\mathrm{hash : X{*} \mapsto 0, 1^n}$$

that a priori uniformly distributes over $\{0,1\}^n$, i.e., each X^* maps to any element in the $\{0,1\}^n$ with probability 2^{-n}.

Then, homomorphism F maps strings in X^* to bit strings in $\{0,1\}^n$ by applying the hash function to the input concatenated with a secret s,

$$Fa := \mathrm{hash(as)}.$$

Note #1: Later, we generalize this to mapping each a in X^* to multiple elements in $\{0,1\}^n$ proportional to $1/P[a]$.

Observe that F is one-way, i.e., there is no homomorphism G such that

$$FGB = A.$$

Theorem 1. *The morphism F defined as*

$$\begin{aligned} X^* &:= hash(a\#s) \\ \text{and} &:= \& \\ \text{or} &:= | \\ \text{complement} &:= \sim \\ \{\} &:= 0^n \\ X^* &:= 1^n. \end{aligned}$$

is a homomorphism.

Proof. The proof is trivial so we omit it. □

Since multiple elements in X^* map to the same element in $\{0,1\}^n$, it is a homomorphism rather than an isomorphism.

What is the probability that two unique elements in X^* map to the same element in $\{0,1\}^n$? That is to say, what is the probability of collision? Since F uniformly distributes over $\{0,1\}^n$, it is just

$$Pr\{x \text{ and } y \text{ collide}\} = 2^(- n).$$

By the law of probability, therefore, the probability that they do not collide is just

$$Pr\{x \text{ and } y \text{ do not collide}\} = 1 - 2^(- n).$$

Next, we define relations on sets. Set membership relation has a characteristic function

```
in : X -> 2^X -> bool
```

which we define as

```
F in a b := a \& b == a.
```

The subset relation has a predicate

```
subset : 2^X -> 2^X -> bool
```

which we define as

```
F subset a b := a \& b == a,
```

just as with the characteristic function, although they have different probabilistic features.

If $X = \{a, b, c\}$, then $2^X = , a, b, c, a, b, a, c, b, c, a, b, c$.

A Boolean index over X is a Boolean algebra over 2^X with $\ = 0$ and $X = 1$ with the normal set operations. This is what a lot of prior work was over.

Note that a type that models `power_set<trapdoor<X>>` is one in which given a value `A` of this type, each element `a` in `A` is a `trapdoor<X>` can be independently observed. This makes it possible to operate on `A` as a normal set, with the exception that the mapping the trapdoors to values may not be obvious (although given a history, or a set of sets, frequency analysis or correlation analysis may reveal quite a bit).

```cpp
template <typename X, size_t N>
struct trapdoor_boolean_algebra
{
    using value_type = X;

    trapdoor_boolean_algebra() :
        value_hash(0),
        key_hash(0)
    {
        // makes the empty set
    }

    trapdoor_boolean_algebra(trapdoor_boolean_algebra const &) = default;

    array<char, N> value_hash;
    array<char, 4> key_hash;
};
```

```cpp
template <typename X, size_t N>
auto make_empty_trapdoor_set()
{
    return trapdoor_boolean_algebra<X,N>();
}
```

```cpp
/**
The disjoint union operation is a partial function that is only defined
when the argument sets are disjoint (it is a dependent type). If they are
not disjoint, the operation has undefined behavior.
 */
template <typename X, size_t N>
auto operator+(
    trapdoor_boolean_algebra<X,N> const & x,
    trapdoor_boolean_algebra<X,N> const & y)
{
    if (x.key_hash != y.key_hash)
        throw invalid_argument("secret key mismatch");

    return trapdoor_boolean_algebra<X>(
        x.value_hash | y.value_hash,
        x.key_hash);
}
```

```cpp
template <typename X, size_t N>
auto operator!(
    trapdoor_boolean_algebra<X,N> const & x)
{
    return trapdoor_boolean_algebra<X>(
        ~x.value_hash,
        x.key_hash);
}
```

```cpp
template <typename X, size_t N>
auto operator*(
    trapdoor_boolean_algebra<X,N> const & x,
    trapdoor_boolean_algebra<X,N> const & y)
```

```
{
    if (x.key_hash != y.key_hash)
        throw invalid_argument("secret key mismatch");

    return trapdoor_boolean_algebra<X>(
        x.value_hash & y.value_hash,
        x.key_hash);
}
```

```
template <typename X, typename Y, size_t N>
auto disjoint_union(
    trapdoor_boolean_algebra<X,N> const & x,
    trapdoor_boolean_algebra<Y,N> const & y)
{
    if (x.key_hash != y.key_hash)
        throw invalid_argument("secret key mismatch");

    return trapdoor_boolean_algebra<variant<X,Y>>(
        x.value_hash | y.value_hash,
        x.key_hash);
}
```

```
// the bernoulli<bool> stores the log-probability of the value being incorrect
template <typename X, size_t N>
bernoulli<bool> empty(trapdoor_boolean_algebra<X,N> const & xs)
{
    auto b = std::all_of(xs.begin(),xs.end(),[](char x) { return x == 0; });
    return bernoulli<bool>{b,0.5};
}

template <typename X, size_t N>
bernoulli<bool> contains(
    trapdoor<X,N> const & x,
    trapdoor_boolean_algebra<X,N> const & xs)
{
    if (x.key_hash != xs.key_hash)
        throw std::invalid_argment("secret key mismatch");
    auto b = std::all_of(xs.begin(),xs.end(),[](char x) { return x == 0; });
    return bernoulli<bool, 1>{b, .5};
}

template <typename X>
approximate_bool operator<=(
    trapdoor_boolean_algebra<X> const & x,
    trapdoor_boolean_algebra<X> const & y)
{
    auto b = std::all_of(xs.begin(),xs.end(),[](char x) { return x == 0; });
    return approximate_bool{b, .5};
}

template <typename X>
approximate_bool operator==(
    trapdoor_boolean_algebra<X> const & x
    trapdoor_boolean_algebra<X> const & y)
```

```cpp
{
    auto b = std::all_of(xs.begin(),xs.end(),[](char x) { return x == 0; });
    return approximate_bool{b, .5};
}


template <typename X, size_t N>
auto hash(trapdoor_boolean_algebra<X,N> const & x)
{
    return x.value_hash ^ x.key_hash ^ hash(typeid(X))
}
```