Lecture Notes for Cryptographic Computing
3. Trusted Dealer Model, Active Security
Message Authentication Codes

Lecturers: Claudio Orlandi, Peter Scholl, Aarhus University

October 23, 2023

1 Before you start

Before reading this note, it is recommended to read the formal definition of security against malicious adversaries (or active security) from [HL10, Sections 2.3]. Here are the references for the material covered this lecture:

1. Information theoretic MACs and how to turn a functionality that gives output to one party to a functionality that gives output to both parties (from [HL10, Sections 2.5.2]);
2. How to add MACs to the OTTT protocol (from [IKMOP13, Section 3.3].).
3. How to add MACs to the BeDOZa protocol (from [NNOB12, Section 3] for the Boolean case and [BDOZ11, Section 3] for the arithmetic case);

Exercise 1. (Mandatory Assignment)

Implement a secure two-party protocol for the *blood type compatibility* function using the passively secure *BeDOZa* protocol and the Boolean formula from the mandatory assignment of the first note. Since the goal of the exercise is to better understand the protocol (not to build a full functioning system), feel free to implement all parties on the same machine and without using network communication. For example, you could implement the dealer, Alice and Bob as three distinct classes and then let them interact in the following way:

```
Dealer.Init();
Alice.Init(x,Dealer.RandA());
Bob.Init(y,Dealer.RandB());
while(Alice.hasOutput()==false)
{
Bob.Receive(Alice.Send());
Alice.Receive(Bob.Send());
}
z = Alice.Output();
```

(Note that the loop is not strictly necessary, since you are implementing the protocol for a specific function so you can predict in advance how many rounds there will be).

Exercise 2. (Active Security is Easy!)

In the first note we recalled that the AND functionality cannot be securely implemented in the presence of passive corruptions. Surprisingly, it is easy to construct a protocol that securely implements the AND functionality against active corruptions (in the case where only Alice learns the output). Do you see why?
Hint 1:
Hint 2:

2 OTTT with Active Security

The OTTT protocol is secure against passive corruptions, as seen in the last lecture. What about active attackers? It is easy to see that a malicious Bob[1] can deviate from the protocol in several ways:

1. *Bob sends the wrong value v':* Instead of sending the value $v = y + s$, Bob sends an arbitrary message $v' \in \{0,1\}^n$. However since Bob knows y and s, this can always be rewritten (from Bob's perspective) as
$$v' = y + s + (v' - y + s) = y + s + err = (y + err) + s = y' + s$$
In the first two steps, we simply rewrite v' as a function of the "real" input y, the randomness from the dealer s and some additive "error factor" $err = (v' - y + s)$. In the last two steps we rewrite v' as an honestly generated message with a different input y'. We conclude that, no matter how Bob chooses v', this can always be rewritten as an honestly generated message with an input y' (possibly different than y). This is a case of *input substitution* and it is tolerated in the setting of active security (the ideal adversary/simulator has the same power in the ideal world).

2. *Bob sends nothing/an invalid message:* this can happen if Bob never sends a message to Alice or equivalently, Bob sends a message pair which is not in the right format i.e., $(v', z'_B) \notin \{0,1\}^n \times \{0,1\}$. Note that the second condition can be efficiently checked by Alice, and the first condition can be checked if we make the assumption that Alice and Bob are connected by a synchronous channel (i.e., Alice has a clock and if she does not receive a message after a predetermined time she decides that Bob has aborted the protocol.).

   Note however that at this stage Bob has learned nothing – the message u received from Alice is a uniformly random string, and Bob never learns the output of the function – and therefore Bob could have made the exact same choice before the protocol starts. In other words, we are not going to consider this as "real" cheating but just as a more convoluted example of *input substitution.*

   To account for this we can simply modify the protocol in such a way that if Alice detects an invalid message (or receives no message), she simply outputs $z = f(x, y_0)$ for some "default" input y_0. This can be efficiently simulated in the ideal world by having the simulator input y_0 to the ideal functionality.

3. *Bob sends a wrong value z'_B:* since z'_B is a bit, if Bob does not send the correct value it must be the case that Bob sends $z'_B = z_B \oplus 1$ (if z'_B is not a bit this will be noticed by Alice as discussed above). This makes the honest Alice output $z' = z \oplus 1 = f(x, y) \oplus 1$.

   Is this "real" cheating, or can this be seen as a case of input substitution too? As a (perhaps artificial) counterexample, consider the case where Alice and Bob are evaluating the function $f(x, y) = 0\ \forall x, y$.

[1] The attack that a malicious Alice can mount are a subset of these.

In the ideal world the simulator cannot find any input which makes Alice output 1. But in the protocol by sending $z'_B = z_B \oplus 1$, Bob makes Alice output 1, thus breaking the correctness of the protocol.[2]

To make the OTTT protocol secure against active adversaries, we must ensure that Bob cannot send the wrong value z_B to Alice. To do so we recall one of the standard tools for checking the integrity of a message.

Message Authentication Code. A MAC scheme has three algorithms $(\mathsf{Gen}, \mathsf{Tag}, \mathsf{Ver})$ where Gen produces a MAC key k, which can be used to compute tags on messages as $t = \mathsf{Tag}(k, x)$. Finally the verification function $\mathsf{Ver}(k, t, x)$ outputs `accept` if t is a valid tag on x with key k or `reject` otherwise. Security of a MAC is defined as a game between a challenger C and an adversary A: C samples a key k, then A is allowed q queries where he can get tags $t_1, \dots, t_q$ on messages $x_1, \dots, x_q$ of his choice. Now the adversary outputs (t', x'). We say that a MAC scheme is $(q, \epsilon)-$secure if no adversary[3] which is allowed $\leq q$ queries can output a pair (t', x') with $x' \neq x_i$ for all i such that t' is a valid MAC on x' with probability $\geq \epsilon$.

A Simple, Information-Theoretic MAC. Consider the following scheme, defined over $\mathbb{Z}_p$ for a large prime p:

- Gen samples random $\alpha, \beta \in \mathbb{Z}_p$ and outputs $k = (\alpha, \beta)$.
- $\mathsf{Tag}(k, x)$: Output the tag $t = \alpha \cdot x + \beta$.
- $\mathsf{Ver}(k, t', x)$: Output `accept` if $t' = \alpha \cdot x + \beta$ or `reject` otherwise.

This MAC is information-theoretically secure, *as long as* (1) Each key is only used once, and (2) the prime p is sufficiently large.

Exercise 3.

Prove that this is a $(1, 1/p)$-secure MAC scheme (even against an unbounded adversary).

Like the one-time pad, this MAC is only one-time secure. To use it in the OTTT protocol we'll need to sample a fresh MAC key for every entry of Bob's matrix. The complete protocol (with differences from the passive protocol highlighted) looks like this:

The dealer: The dealer D performs the following operations:

1. Choose two shifts $r \in \{0,1\}^n$ and $s \in \{0,1\}^n$ uniformly at random;
2. Choose a matrix $M_B \in \{0,1\}^{2^n \times 2^n}$ uniformly at random;
3. Compute a matrix M_A such that $M_A[i,j] = M_B[i,j] \oplus f(i - r \bmod 2^n, j - s \bmod 2^n)$;
4. Generate 2^{2n} keys for a $(1, \epsilon)$-secure MAC i.e., $\forall i, j \in \{0,1\}^n$ define $K[i,j] \leftarrow \mathsf{Gen}()$;

[2] Of course, in the specific counterexample Alice could check that the output is invalid and call Bob a cheater. One might be tempted to think that it is always possible to have Alice check that the output of the function is "valid" in some sense, and thus prevent cheating. For instance, Alice could check that the output is in the range of the function. This does not work in general: it is not always possible to efficiently check if the output of a function is in its range (think of a pseudo-random generator which expands from n to $2n$ bits), and even more importantly this would not suffice for our definition of security. The definition of active security we are working with does not just require that *there exist a y such that* $z = f(x, y)$, but that *Bob knows a y such that* $z = f(x, y)$. Note that the "existential" definition of correctness would consider the following protocol to be secure/correct 1) Bob sends Alice a function $g : \{0,1\}^n \to \{0,1\}^n$ and 2) Alice outputs $z = f(x, g(x))$. On the other hand, such a protocol is not considered secure/correct by our definition of security (the issue describe above is commonly known as *input independence*).

[3] Most MACs you have seen so far are secure only against computationally bounded adversaries. However in this notes we will use MACs which are secure even against unbounded adversaries.

5. Generate MACs for all values in M_B i.e., $\forall i,j \in \{0,1\}^n$ define $T[i,j] \leftarrow \mathsf{Tag}(K[i,j], M_B[i,j])$;
6. Output (r, M_A, K) to Alice and (s, M_B, T) to Bob;

The protocol: Having received (r, M_A, K) and (s, M_B, T) from D, A and B with input x and y:

1. Alice computes $u = x + r \bmod 2^n$ and sends it to Bob;
2. Bob computes $v = y + s \bmod 2^n$, $z_B = M_B[u,v]$ and $t_B = T[u,v]$ and sends (v, z_B, t_B) to Alice;
3. If $\mathsf{Ver}(K[u,v], t_B, z_B) = \texttt{reject}$ (or no valid message is received) Alice outputs $z = f(x, y_0)$; else Alice outputs $z = M_A[u,v] \oplus z_B$;

To prove that this protocol is secure against a corrupted Bob, we need to construct a simulator, and prove that the *joint distribution* of the view produced by the simulator together with the output of Alice in the ideal world is indistinguishable from the view of a corrupted Bob and the output of Alice in the protocol execution.

Our statement is of the form: *assuming that the MAC scheme is secure, then the enhanced OTTT protocol is secure*. As usual in proofs for statements in this form, we will assume that if the protocol is insecure (i.e., if there exists an adversary that can distinguish between the real world and the ideal world) then we can construct an adversary that breaks the security of the MAC scheme.

The simulator works in the following way: 1) sample random s, M_B, generate keys K for the MAC scheme and compute MACs $T = \mathsf{Tag}(K, M_B)$ and send (s, M_B, T) to Bob (replacing the trusted dealer); 2) sample a random u and send it to Bob (replacing the honest Alice); 3) If Bob does not output anything, or outputs an invalid message, or outputs a triple (v', z'_B, t'_B) such that $z'_B \neq M_B[u,v]$ or $\mathsf{Ver}(K[u,v], z'_B, t'_B) = \texttt{reject}$ the simulator inputs y_0 to the ideal functionality. Else, the simulator inputs $y' = v' - s$ to the ideal functionality.

We now need to analyse the distribution produced in this mental experiment, and compare it with the distribution in the real world: *the view of Bob* is distributed identically in the simulation and in the real protocol: M_B, r are chosen uniformly at random in both experiments, while u is computed as $x + r$ (with random r) in the real protocol and as a uniform random value in the simulation, leading to the same distribution. On the other hand *the output of Alice* is identically distributed in the two worlds except for the single case where the corrupted Bob outputs a triple such that $\mathsf{Ver}(K[u,v], t'_B, z'_B) = \texttt{accept}$ and $z'_B \neq M_B[u,v]$: when this happens in the real world Alice outputs $z' = M_A \oplus z'_B = f(x, y') \oplus 1$ while in the ideal world Alice outputs $z' = f(x, y_0)$. However such a corrupted Bob can trivially be turned into an adversary for the MAC scheme, and this concludes the proof.

3 BeDoZa with MACs

The BeDOZa protocol also suffers from the problem that when the command OpenTo is invoked, a corrupted party can lie about his/her share. This means that an active adversary can add arbitrary values to any of the internal values of the circuit (those attacks are therefore called "additive attacks" in the literature). Note that in this case an attack against correctness might turn into an attack against privacy, since the attacker can essentially change which function is being evaluated. Consider the following, simple example: Alice and Bob are computing the function:

$$f(x, y) := (x \oplus 1) \cdot x$$

That is a function that ignores Bob's input, and evaluates a simple circuit in the variable x. Note that the output of the function is always 0, regardless of the inputs. Thus, a corrupt Bob should not learn any information about the input of Alice. But, in a setting where additive attacks are present, Bob might be able to "add one" to the left wire of the multiplication gate. That is, Bob can change the function to be computed to

$$\tilde{f}(x, y) := ((x \oplus 1) \oplus 1) \cdot x$$

Note that $\tilde{f}(x, y) = x$, therefore an attack on the correctness of the protocol turns into an attack against privacy.

We can fix this using MACs, but we need to do so in a way that allows to perform linear operations "for free" on shared value. Ideally we would like a *homomorphic MAC*, where given two tags t_1, t_2 on two values x_1, x_2 (but not the key k), anyone can compute a MAC on any linear combination of x_1, x_2.

For instance, we might want a MAC scheme where

$$t' = a \cdot t_1 + b \cdot t_2$$

is a valid MAC for the message

$$x' = a \cdot x_1 + b \cdot x_2$$

One problem with such a MAC is that it now seems like the adversary can forge MACs on values of his choice, using the homomorphism! To avoid this issue, we require the *verifier* to also apply a homomorphic operation on its key, so that it can verify the correct operation was applied to the message.

Building Homomorphic m-time MACs. We can easily tweak the one-time MAC construction to allow for linear homomorphism. The Gen algorithm outputs keys $k_1, \ldots, k_m$ where $k_i = (\alpha, \beta_i)$ that is, the first part stays constant while the second part of the key is different for each value to be MACed. Now we can compute $t_i = \mathsf{Tag}(k_i, x_i)$ as follows:

$$t_i = \alpha \cdot x_i + \beta_i$$

Now if one has two MACs t_1, t_2 for values x_1, x_2, it is possible to compute a MAC $t' = t_1 + t_2$ which is a valid MAC for $x' = x_1 + x_2$ *under the key* $k' = (\alpha, \beta_1 + \beta_2)$.

❓ Exercise 4.

Prove that having a fixed α does not break the security. That is, think of the following security game: the adversary can query the challenger on messages $x_1, \ldots, x_q$ of his choice and every time receives $t_i \leftarrow \mathsf{Tag}(k_i, x_i)$ for a newly generated $k_i \leftarrow \mathsf{Gen}$ where $k_i = (\alpha, \beta_i)$ for all i. Now the adversary outputs a tuple (i, t', x') and wins if $x' \neq x_i$ and $\mathsf{Ver}(k_i, t', x')$. Prove that no adversary wins this game with probability greater than $1/p$.

Given these MACs, it is possible to modify the BeDOZa protocol to be secure also against actively corrupted parties. We do so by adding MACs to the "shared representation" $[x]$. We will look at the arithmetic version of BeDOZa where all values are taken from $\mathbb{Z}_p$ for a large p. Afterwards we discuss how to cope with Boolean circuits.

Invariant: Given a value $x \in \mathbb{Z}_p$, we now write $[x]$ to denote the following situation:

- Alice knows x_A, Bob knows x_B such that $x = x_A + x_B$;
- Alice knows a MAC key $k_{A,x} = (\alpha_A, \beta_{A,x})$, Bob knows a MAC key $k_{B,x} = (\alpha_B, \beta_{B,x})$; Note that α_A, α_B is constant for the whole circuit;
- Alice knows a tag $t_{A,x} = \mathsf{Tag}(k_{B,x}, x_A)$, Bob knows a tag $t_{B,x} = \mathsf{Tag}(k_{A,x}, x_B)$;

Output Wires: If Alice (resp. Bob) is supposed to learn a secret value $[x]$, Bob sends x_B to Alice together with $t_{B,x}$. Alice outputs $x = x_A + x_B$ if $\mathsf{Ver}(k_{A,x}, t_{B,x}, x_B)$. (Otherwise, Alice aborts the protocol and outputs $z = f(x, y_0)$, whereas Bob aborts and does nothing).

We write $(x, \perp) \leftarrow \text{OpenTo}(A, [x])$. If both Alice and Bob are supposed to learn an output, we write $x \leftarrow \text{OpenToAll}([x])$ as a shortcut for $(x, \perp) \leftarrow \text{OpenTo}(A, [x])$ and $(\perp, x) \leftarrow \text{OpenTo}(B, [x])$.

Addition of Two Wires: If Alice and Bob have two secret shared values $[x], [y]$ and want to compute a secret shared representation of $z = x + y$ they can simply compute $z_A = x_A + y_A$ and $z_B = x_B + y_B$. They also compute MACs

$$t_{A,z} = t_{A,x} + t_{A,y} \text{ and } t_{B,z} = t_{B,x} + t_{B,y}$$

and new MAC keys

$$k_{A,z} = (\alpha_A, \beta_{A,x} + \beta_{A,y}) \text{ and } k_{B,z} = (\alpha_B, \beta_{B,x} + \beta_{B,y})$$

We write $[z] = [x] + [y]$ to denote this sub protocol.

Input Wires: For each of the n wires belonging to Alice, both parties run the following protocol (the case for Bob is symmetric):

1. The dealer D outputs a random share (with MACs) $[r]$;
2. Alice and Bob run $(r, \perp) \leftarrow \text{OpenTo}(A, [r])$;
3. Alice sends Bob $d = x - r$;
4. Alice and Bob compute $[x] = [r] + d$;

Note that in the protocol Alice and Bob do not generate their own MAC keys, and they never use the Tag function. In other words, all keys and MACs are generated by the trusted dealer (either as parts of the random shares which are sent out in the input subprotocol or as parts of the random multiplicative triples which are sent out in the multiplication subprotocol – see exercise below). In the whole protocol Alice and Bob only compute linear combinations of shares, MACs and keys and verify the correctness of received MACs.

❓ **Exercise 5.** (Addition with Constant, Multiplication with Constant)

Design subprotocols for addition with constant and multiplication with constant with MACs, and argue for their correctness/security.
Hint 1 (for addition):

Hint 2:
Hint 3 (for multiplication):

❓ **Exercise 6.** (Multiplication of Two Wires)

Note that we did not describe the evaluation of multiplication gates: The evaluation of a multiplication gate is done exactly as in the passive secure BeDOZa protocol, except that the triple $[u], [v], [w]$ received from the trusted dealer also contains MACs and keys. Argue that this works as expected.

Security. We only describe the intuition behind the proof of security of this protocol. First of all, note that the input phase is very different from the passively secure version of the protocol: here we ask the dealer for a random value $[r]$ which is then revealed to Alice and used to masked her input. This step might seem unnecessary, but it is actually crucial to prove security. As argued before for the OTTT protocol, to be able to prove security against active adversaries the simulator must be able to *extract* an input from the corrupted party which can then be input to the ideal functionality. The new input subprotocol allows exactly that: in the simulation the simulator will choose the random value r and will therefore be able to extract Alice's input as $x' = r + d$. All internal gates are simulated essentially in the same way as it was done for the passively secure case, but in addition the simulator will also verify that Bob sends the values he is supposed to and abort otherwise. The resulting views and output distributions will be indistinguishable, and this can be argued in a similar way as what was done for the OTTT protocol.

Efficiency. The BeDOZa protocol enhanced with MACs scheme guarantees that no adversary can cheat with probability significantly bigger than $O(1/p)$. This requires that p must be at least $40-60$ bits long to give any meaningful security guarantee. When this is the case the protocol only has a constant overhead compared with the passively secure version (for each wire carrying a value in $\mathbb{Z}_p$ a party needs to store 3 values in $\mathbb{Z}_p$). What if we want to evaluate a Boolean circuit (i.e., $p=2$) or a circuit over some other small field? The simple solution is to have k MACs on each value, with k such that $1/p^k$ is small enough. This is what is done in the TinyOT protocol [NNOB12]. Unfortunately this increases the storage overhead to $O(k)$. In the MiniMAC protocol [DZ13, DLT14], vectors of bits are MACed together and therefore we can again achieve (amortized) overhead $O(1)$.[4] The main problem solved in this paper is how to make vector MACs which are still homomorphic.

A different problem arises when there are more than 2 parties (it is not hard to see that the BeDOZa protocol, both in its passively and actively secure variant, can be generalized to more than 2 parties). In this case, each party has a share x_i s.t., $\sum x_i = x$. If we extend the protocol described here to n parties, then each party needs to have a key and a MAC for each other party (this is what is done in the BeDOZa protocol [BDOZ11]), leading to an $O(n)$ storage overhead for each party. A different approach is taken in the SPeeDZ protocol [DPSZ12, DKLPSS13]. Here, instead of putting MACs on shares, MACs are computed directly on the values, and then *the MACs and the keys are secret shared!* In other words, in SPeeDZ each party only stores 3 values (x_i, k_i, t_i) such that $\sum t_i$ is a valid MAC on $\sum x_i$ with key $\sum k_i$. This, however, introduces other problems which are solved by the mentioned papers in a very clever way![5]

4 Other Exercises

❓ **Exercise 7.** (Multi-party Output, Active Security)

We saw how to use MACs to force an actively corrupted party to send the right output (or nothing) to the other party. Let's try to generalize this to the multiparty case – for instance 3. The protocol that we want to implement is very simple: Alice receives the output of the function z from the ideal functionality and has to forward it to Bob and Charlie. We are worried that Alice might cheat and send a different message z' instead.

1. The first attempt would be simply to have both Bob and Charlie to choose a key for the MAC, so to say Alice inputs x_A, Bob and Charlie input resp. (x_B, k_B) and (x_C, k_C) to the functionality that output $z = f(x_A, x_B, x_C)$ together with two MACs $t_B = \mathsf{Tag}(k_B, z)$ and $t_C = \mathsf{Tag}(k_C, z)$ to Alice.

   Now Alice sends (z'_B, t'_B) to Bob who outputs z'_B if $\mathsf{Ver}(k_B, z'_B, t'_B) = \texttt{accept}$ or $\perp$ otherwise. In the same way, Charlie outputs z'_C if $\mathsf{Ver}(k_C, z'_C, t'_C) = \texttt{accept}$ or $\perp$ otherwise. Can Alice cheat? (where by cheat we mean something that she could not have done in the ideal scenario where there is a trusted party that computes the function, gives her the output, and then asks her for permission to deliver the output to all honest parties?)

   Hint :

2. If you found an attack above, could you fix the problem if Alice was forced to send the same message to Bob and Charlie? Would it help if Bob and Charlie could talk to each other?

3. Here is a different MAC scheme which might help solving the above problem: now Alice inputs x_A to the ideal functionality, while Bob and Charlie input respectively (x_B, α_B, β_B) and (x_C, α_C, β_C). The ideal functionality computes $z = f(x_A, x_B, x_C)$ and three values $h_0, h_1, h_2 \in \mathbb{Z}_p$ which define a degree 2 polynomial $h(x) = h_2x^2 + h_1x + h_0 \mod p$ such that s.t., $h(0) = z$, $h(\alpha_B) = \beta_B$

[4] This could be a good topic for your project!

[5] This is also a good topic for your project!

and $h(\alpha_C) = \beta_C$. Finally Alice sends (h'_2, h'_1, h'_0) to both Bob and Charlie (assume some magic mechanism which prevents Alice from sending different messages to the two parties) who output $z_B = h'_0$ if and only if $h'(\alpha_B) = \beta_B$ and $z_C = h'_0$ if and only if $h'(\alpha_C) = \beta_C$ (where h' is the polynomial defined by (h'_2, h'_1, h'_0)). Prove that no matter how a corrupted Alice chooses (h'_2, h'_1, h'_0) the output z'_B, z'_C of Bob and Charlie is either $(z'_B, z'_C) = (z, z)$ or $(z'_B, z'_C) = (\perp, \perp)$ – in other words, no malicious Alice can make Bob and Charlie output different values or incorrect values.

❓ **Exercise 8.** (Perfect Security – Hard!)

Using MACs together with OTTT "only" gets statistical security (meaning that is a probability, even if very small, that the adversary can forge the MACs).
It is actually possible to modify the OTTT protocol is such a way that the protocol guarantees perfect security. Interestingly, the modified protocol is not only more secure, but more efficient as well.

1. Start by looking at the protocol for equality from [IKMOP13, Figure 1] Can you prove that this protocol is secure against active adversaries as well?

   Hint :

2. Now look at [IKMOP13, Figure 4] (described below as well) – in some sense this protocol generalizes the idea of the equality protocol to any function. Can you prove its security?

The dealer: The dealer D performs the following operations:

1. Choose a shift $r \in \{0,1\}^n$;
2. Choose a matrix $P[i,j] \in (\{0,1\}^n)^{2^n \times 2^n}$ where each row represents a different random permutation of $\{0,1\}^n$. In other words, for all $i \in \{0,1\}^n$, pick a random permutation π_i and set $P[i,j] = \pi_i(j)$ for all j;
3. Define a matrix $M \in \{0,1\}^{2^n \times 2^n}$ in the following way: for all $i, j \in \{0,1\}^n$

$$M[i, P[i+r, j]] := f(i,j)$$

4. Output (r, M) to Alice and P to Bob;

The protocol: Having received (r, M) and P from D, A and B with input x and y:

1. Alice computes $u = x + r \bmod 2^n$ and sends it to Bob;
2. Bob computes $y_B = P[u, y]$ and sends it to Alice;
3. If no valid message is received Alice outputs $z = f(x, y_0)$; else Alice outputs $z = M[x, y_B]$;

References

[HL10] Carmit Hazay, Yehuda Lindell
Efficient Secure Two-Party Protocols
`http://lib.myilibrary.com.ez.statsbiblioteket.dk:2048/Open.aspx?id=300348`

[BDOZ11] Rikke Bendlin, Ivan Damgård, Claudio Orlandi and Sarah Zakarias.
Semi-Homomorphic Encryption and Multiparty Computation
EUROCRYPT 2011.
Available `http://eprint.iacr.org/2010/514`

[NNOB12] Jesper Buus Nielsen, Peter Sebastian Nordholt, Claudio Orlandi, Sai Sheshank.
A New Approach to Practical Active-Secure Two-Party Computation
CRYPTO 2012.
Available `http://eprint.iacr.org/2011/091`

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel Smart, Sarah Zakarias
Multiparty Computation from Somewhat Homomorphic Encryption
CRYPTO 2012.
Available `http://eprint.iacr.org/2011/535`

[DKLPSS13] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, Nigel Smart,
Practical Covertly Secure MPC for Dishonest Majority - Or: Breaking the SPDZ Limits
ESORICS 2013.
Available `http://eprint.iacr.org/2012/642`

[IKMOP13] Yuval Ishai, Eyal Kushilevitz, Claudio Orlandi, Sigurd Meldgaard and Anat Paskin.
On the Power Of Correlated Randomness for Secure Computation.
TCC 2013.
Available `http://cs.au.dk/~orlandi/tcc2013-draft.pdf`

[DZ13] Ivan Damgård, Sarah Zakarias
Constant-Overhead Secure Computation of Boolean Circuits using Preprocessing
TCC 2013.
Available `http://eprint.iacr.org/2012/512`

[DLT14] Ivan Damgård, Rasmus Lauritsen, Tomas Toft
An Empirical Study and some Improvements of the MiniMac Protocol for Secure Computation
SCN 2014.
Available `http://eprint.iacr.org/2014/289`