Demo: High-Throughput Secure Three-Party Computation of Kerberos Ticket Generation

Toshinori Araki
NEC Corporation, Japan
t-araki@ek.jp.nec.com

Assaf Barak
Bar-Ilan University, Israel
assaf.barak@biu.ac.il

Jun Furukawa
NEC Corporation, Japan
j-furukawa@ay.jp.nec.com

Yehuda Lindell*
Dept. of Computer Science
Bar-Ilan University, Israel
lindell@biu.ac.il

Ariel Nof*
Dept. of Computer Science
Bar-Ilan University, Israel
nofdinar@gmail.com

Kazuma Ohara
NEC Corporation, Japan
k-ohara@ax.jp.nec.com

ABSTRACT

Secure multi-party computation (SMPC) is a cryptographic tool that enables a set of parties to jointly compute any function of their inputs while keeping the privacy of inputs. The paper "High Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority" in this ACM CCS 2016 [4] presents a new protocol which its implementation carried out over 1,300,000 AESs per second and was able to support 35,000 login queries of Kerberos authentication per second. This poster/demo presents the design of the implementation and demonstrates the Kerberos authentication over here. The design will show how this high-throughput three-party computation can be done using simple servers. The demonstration proves that secure multiparty computation of Kerberos authentications in large organizations is now practical.

1. INTRODUCTION

The authors of this poster proposed, in their paper [4] at this ACM CCS 2016, a novel provably secure three-party computation protocol with honest majority. This paper focused on *the throughput* of multiparty computation and achieved over 1,300,000 AES encryptions per second, which is 14 times faster than the latest optimized version [3] of Sharemind protocol. The paper also reports that, under this capability, 35,000 login queries of Kerberos authentication were processed per second.

In this poster, we are going to present the detail design of our implementation which enabled this high throughput and demonstrate the Kerberos authentication application.

*Supported by the European Research Council under the ERC consolidators grant agreement n. 615172 (HIPS) and under the European Union's Seventh Framework Program (FP7/2007-2013) grant agreement n. 609611 (PRACTICE), and by the BIU Center for Research in Applied Cryptography and Cyber Security in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office.

CCS'16, October 24-28, 2016, Vienna, Austria

ACM ISBN 978-1-4503-4139-4/16/10.

DOI: http://dx.doi.org/10.1145/2976749.2989035

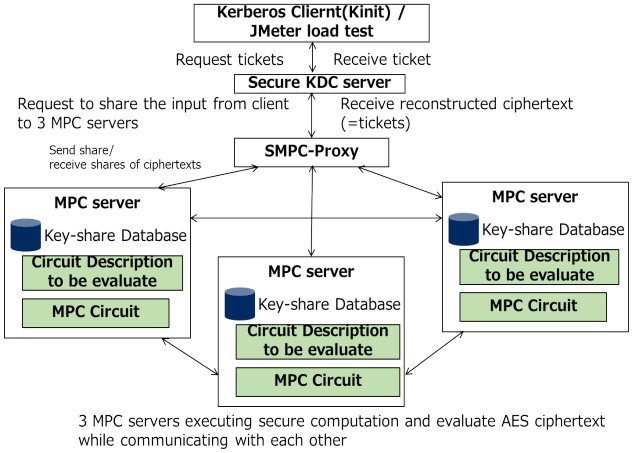

Figure 1: The Kerberos authentication using MPC

Although the protocol presented in the paper itself requires small communication, it does not immediately imply high throughput. Our design of the implementation leveraged the simple structure of the protocol so as to continuously produce the stream of bits necessary to carry out the high throughput multiparty computation. We also show that this design is suitable for being applied to Kerberos authentication. We present how the stream of login queries are converted to the stream of multiparty computation and then re-converted to the stream of responses to the login queries. This description of design and the demo complement our paper at this ACM CCS 2016.

2. DESIGN FOR PARALLEL AES

We have implemented our SMPC protocol and applied it to Kerberos authentication server, which issues Ticket-Granting-Ticket (TGT) by AES encryption. The system view is shown in Figure 1. The "Secure KDC server" is the Kerberos server which accepts requests of issuing tickets from the clients. This secure KDC server generates tickets with the help of new SMPC-proxy process within that server instead of simply calling AES encryption functionality. According to our design, although the server sends a request to the SMPC-proxy for each encryption, it does not receive the encrypted messages one-by-one. Instead, it receives a

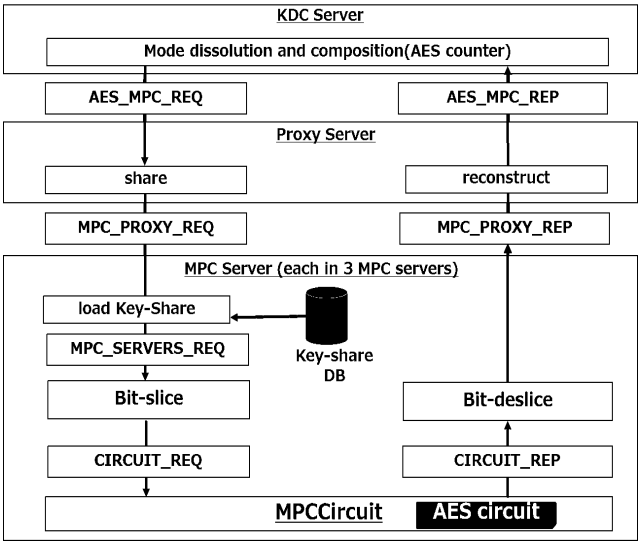

Figure 2: The data flow of our inplementation

batch of encrypted messages only after a while. Hence the encryptions are done in parallel for multiple messages. We have modified Kerberos server to process ticket generation in such a way.

The SMPC-proxy keeps receiving requests for encrypting message for a certain small period of time. We consider only the AES counter mode, and hence each request is a pair of principal identifier and counter. After the proxy received a sequence of these messages, it generates shares of them, and sends each share to the corresponding MPC server.

Each of three MPC servers, upon receiving a request from the SMPC-proxy, first loads the share of the key of the specified principal. The shared key is that of the expanded round keys. Then, to achieve very high throughput, MPC servers process the conversion of the data structure which is called as "bit-slice" (described in Sect. 2.2). MPC server applies bit-slice the shared keys and counters. These bit-sliced counters are encrypted by bit-sliced keys according to our SMPC protocol. Finally, the obtained bit-sliced ciphertexts are bit-desliced to a sequence of ciphertexts and returned to the SMPC-proxy. SMPC-proxy now returns the results to the KDC server, which assembles them to generate tickets.

2.1 Architecture

Our system is composed of KDC server, SMPC-proxy, MPC servers, and clients. Each component is implemented as in the following.

KDC server It works as a KDC server for Kerberos 5. The encryption mode of operation for ticket generation is 128-bit AES-CTR, where all principal keys are secret-shared by 3 MPC servers. The KDC server has a new function that, when received a pair of principal identity and message, dissolves into a sequence of pairs of the same principal identifier and 128 bit counter. This function converts a request for encrypting a message of arbitrary size into a sequence of requests for encrypting a 128 bit blocks. This sequence is implemented in the following structure and transferred to SMPC-proxy.

```
struct AES_MPC_REQ {
  uint32   numAES;
  uint32  *principalID;
  uint32  *counter; //4 unit32 for one counter.
};
```

Note that a sequence of the same principal identifier may appears in `*principalID` if they are from the same request for encrypting a message.

The result is returned by structure `AES_MPC_REQ` which is defined similarly without principal identifier. This will be reconstructed into ciphertexts of AES-counter mode.

SMPC-Proxy SMPC-proxy is implemented by `Class SMPCProxy`. Its major methods are simply, to be given data (`struct AES_MPC_REQ`) from KDC service, converts data into shares, sending these shares (`struct MPC_PROXY_REQ`) to MPC servers, receiving the shares (`struct MPC_PROXY_REP`) of ciphertexts from MPC servers, recover the ciphertexts from shares, and returning ciphertexts (`struct AES_MPC_REQ`) to KDC server.

MPC server MPC servers are implemented by `Class MPCServer{}`. Each MPC server receives `struct MPC_Proxy_REQ` which is defined as

```
struct MPC_Proxy_REQ {
  uint32 numAES;
  uint32 *prncipalID;
  struct SharedPlain plMsgShare; //shares
};
```

MPC server loads the shares of the expanded secret key of each principal by method `LoadKeyShare`. The result is `struct MPC_SERVERS_REQ` defined below.

```
struct MPC_SERVERS_REQ {
  uint32 numAES;
  struct SharedPlain plKeyShare;
  struct SharedPlain plMsgShare;
};
```

As the number of shares of both secret keys and messages are the same and they are ordered in the same manner, we can apply bit-slicing theme sequentially to obtain `struct CIRCUIT_REQ`.

```
struct CIRCUIT_REQ {
  uint32 numAES;
  /* bit-sliced input */
  struct SharedSlices slMsgShare;
};
```

Note that there is only one array in this structure. This sliced data will be given to `Class MPCCircuit` to MPC. The result of this MPC is sent to SMPC-Proxy. How bit-slicing and bit-deslicing is implemented is described later.

An instance of `Class MPCCircuit` is constructed by giving a circuit description. In our case it is of AES. This class is general in that it runs MPC of any function in bit-sliced manner if its appropriate description is given.

Client Nothing is special about our clients except we require them to use AES-counter mode.

2.2 SIMD Bit-Slicing and PRNG

To highly parallelize MPC for high throughput, we used the technique called by "bit-slice". The detail of bit-slice procedure is described in our paper [4]. The bit-sliced data structure allows us to compute multiple operations for bit value as the operation for one vector, and it is compatible with Intel SIMD instructions. We assumed that the number of plain data is very large in case of our application since it corresponds to the number of users. This paper shows more detail about the data structure and how to apply bit-slicing/SIMD technique for our protocol, which is omitted from the conference paper [4].

XOR/AND operation for bit-sliced data

The computation/communication costs of all MPC instruction used in our protocol is very simple and light. For 1-bit secret information, each party has 2-bit as a share of 1-bit. 1-bit XOR instruction for MPC requires 2-bit (ordinary) XOR operation and no communication, and 1-bit AND instruction requires 2-bit (ordinary) AND operation, 3-bit XOR operation, 1-bit random number generation and 1-bit communication with other parties (For the detail of the protocol, see the proceedings of CCS'16 [4]).

In our protocol, the *s*liced vector m'_i for the inputs consist of i-th bit's shares of all inputs which are parallely entered, as like follows

$$m'_i = (s_{0,0}, s_{1,0}, \ldots, s_{l,0} || s_{0,1}, s_{1,1}, \ldots, s_{l,1}) \ (i \in [n])$$

where n corresponds to the index of wires of the circuit, and l corresponds to the number of parallel inputs. Each of the share of secret bit is represented as $(s_{0,k}, s_{1,k})$, i.e., the former half bits of m'_i are the first elements of the shares, and latter half is second elements of the shares.

If we want to evaluate XOR/AND instruction for the i-th bit and the j-th bit by MPC, we need to compute XOR/AND for the shares of i-th and j-th bit. It can be realized by the *v*ector-wise XOR/AND operation of m'_i and m'_j, and this operation can be made more faster by SIMD instruction.

In addition, for the random number generation used in AND instruction of MPC, we use AES counter mode by AES-NI, which has very useful property that we can use it to perform up to 8 encryptions in one instruction.

We stress that this optimization is very effective since the local XOR/AND gate computations take possession of 49.82% of whole computation time (see also [4]).

2.3 AES Circuit

We made our AES circuit description based on the AES circuit suitable for MPC [5]. This description has no delimiters of round. So, we add the delimiters. The policy of adding delimiters is simple. When no more gates compute without finishing AND computation, a round change delimiter is add to that point. The properties of derived circuit are summarized as follows.

Table 1: The properties of our circuit

The number of round	40
The number of gate	27692
The max number of AND gate in a round	288
The min number of AND gate in a round	48

In this description, a gate is represented by following contents.

- Number of input
- Number of output
- Indexes of input wires
- Index of output wire
- Gate type (XOR = 1, AND = 2, INV = 3)
- Index of buffer used for sending data. (−1 means the outout doesn't be sent.)

Some example of our circuit description is as follows. The delimiter of round is represented by 000000.

```
2 1 28491 28258 28383 1 -1 // XOR gate
1 1 28240 28457 3 -1 // INV gate
2 1 28512 28513 28494 2 32 // AND gate
0 0 0 0 0 0 // Change round delimiter
```

3. CONTENTS OF THE DEMO

We will demonstrate our Kerberos authentication server by showing the performance of it in real environment. The hardware construction of the experiments is 3 server machines where each has two Intel Xeon E5-2650 v3 2.3GHz CPUs with a total of 20 cores, and these are connected via 10Gbps LAN.

First, we simulate the three party computation for issuing Kerberos TGT tickets by these server machines where the JMeter load test tool simulates clients. You can see the detailed performance of our implementation in this demonstration, for example on the following items.

- Throughput
- Latency
- Rate of CPU utilization
- Rate of network utilization

This experiments will be done while changing the parameters, i.e., the number of cores and the number of clients.

Second, we also demonstrate with the client as Kinit which is a command for obtaining/renewing TGT ticket in an ordinary Kerberos system. These experiments show that the users of our implementation can be authenticated by same manner as usual Kerberos authentication. It is very important feature since the usability of the system for users does not compromised and it lowers the barrier of using this system.

4. REFERENCES

[1] D. Bogdanov, S. Laur and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In *ESORICS 2008*, Springer (LNCS 5283), 192–206, 2008.

[2] D. Bogdanov, M. Niitsoo, T. Toft, J. Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec. 11(6):* 403-418, 2012.

[3] L. Kerik, P. Laud and J. Randmets. Optimizing MPC for robust and scalable integer and floating-point arithmetic. In *4th WAHC*, 2016.

[4] T. Araki, J. Furukara, Y .Lindell, A. Nof, K. Ohara. *High-Throughput Semi-Honest Secure Three-Party Computation with an Honest Majority.* In ACM CCS 2016, to be appeared, 2016.

[5] "Circuits of Basic Functions Suitable For MPC and FHE." http://www.cs.bris.ac.uk/Research/CryptographySecurity/MPC/

[6] "Intel Architecture Instruction Set Extensions Programming Reference." http://www.naic.edu/˜phil/software/intel/319433-014.pdf