Privacy-Preserving HIE by Multi-Party Deterministic Noise Generation (Appendix)

Yuzhe Tang
Syracuse University
Syracuse, NY USA
ytang100@syr.edu

Xi Liu
Georgia Tech
Atlanta, GA USA
xliu314@gatech.edu

Katchaguy Areekijseree
Syracuse University
Syracuse, NY USA
kareekij@syr.edu

1. REFERENCES

[1] S. G. Choi, J. Katz, R. Kumaresan, and H.-S. Zhou. On the security of the free-xor technique. In *Theory of Cryptography*, pages 39–53. Springer, 2012.

[2] J. Seward, N. Nethercote, and J. Weidendorfer. *Valgrind 3.3-Advanced Debugging and Profiling for GNU/Linux applications*. Network Theory Ltd., 2008.

APPENDIX

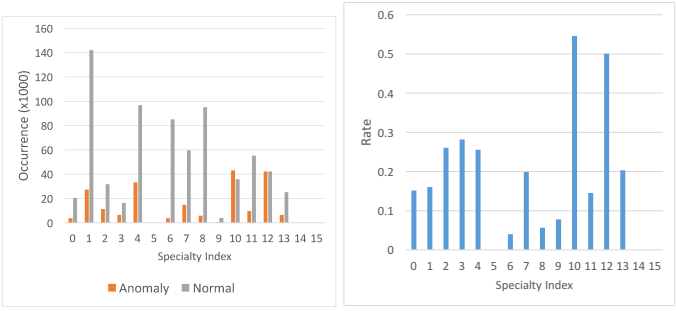
A. APPENDIX: EXTRA EXPERIMENTS

Table 1: Hospital ranking in specialties (from US-NEWS)

ID.	Specialty ranking															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
1	1	1	0	0	1	1	1	1	1	0	1	0	1	0	0	1
2	0	0	1	0	1	1	1	1	1	0	1	0	1	0	0	0
3	1	1	1	0	1	1	1	1	2	0	1	0	1	0	0	1
4	1	2	0	0	2	2	0	1	2	0	2	0	1	0	0	1
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
6	0	0	1	0	1	1	0	1	1	0	0	0	0	0	0	1
7	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
8	0	0	1	1	1	1	0	1	1	0	1	0	1	0	0	1
9	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
11	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	1	1	0	1	0	1	0	0	0	0	1
13	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
14	1	0	0	0	1	0	0	1	0	0	1	0	1	0	0	0
15	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
16	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0	1
17	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0	1
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
19	1	0	0	0	0	1	1	0	1	0	0	0	1	0	0	1
20	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
21	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
22	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0
23	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
24	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0
25	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
26	0	0	0	0	0	2	0	0	0	0	2	0	0	0	0	0
27	1	2	1	1	1	1	1	1	1	0	1	0	1	0	0	1
28	2	0	0	2	1	1	2	0	0	0	0	0	0	0	0	2
29	2	2	2	2	2	2	1	2	2	0	1	1	1	2	0	2
30	2	2	2	2	2	2	2	2	2	1	2	2	2	2	2	2
31	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0
32	2	2	2	2	2	2	2	2	2	1	2	2	2	2	2	2
33	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	1
34	0	0	0	1	0	1	0	1	1	0	0	0	0	0	0	1
35	1	1	0	0	0	1	0	1	1	0	0	1	0	0	0	0
36	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1
37	0	0	0	0	0	1	0	1	1	0	0	0	0	0	0	1
38	1	1	1	0	1	1	0	1	1	0	1	0	1	0	0	1
39	2	2	2	2	2	2	1	2	2	0	2	0	2	2	2	2

A.1 Filtering out anomaly

We pre-filter the dataset to concentrate our experiments on the part of dataset that needs protection. Specifically, in our P^{3}I system, the part of data that needs protection is the one that are in line with the assumption: "A patient tends to visit a hospital specialized in treating the disease the patient has." This is the common knowledge known to the attacker and the ground on which the attacker mounts her attacks. The anomaly rate in the raw dataset then measures the probability at which the data does not meet the assumption. We measure the anomaly rate by counting the occurrence of the cases that the hospital specialty does not match the patient's diagnose code in the Sparcs dataset. Based on the empirical result (shown in Figure 1 in the Appendix), the anomaly rate is low for certain specialties (e.g., Cancer (0), Ophthalmology (9) and Rehabilition (13)) and high for other specialties (e.g., Gastroenterology & GI Surgery (4), Orthopedic (10) and Pulmonology (12)). Our experiments focus on the low-anomaly-rate cases in line with our assumption.



(a) Count (b) Rate

Figure 1: Anomaly in dataset

A.2 Specialization

We study the relationship between attack success rate and the degree of specialization of a hospital. The intuition is that visiting a specialized clinic (e.g., woman's center) should disclose more information, thus being more vulnerable, than visiting a general hospital. The observation guides the design of our P^{3}I construction in that P^{3}I aims at expected diversity l instead of k, and naturally injects more noises for highly specialized hospitals.

The specialization degree of a hospital is measured by the sum of the inverse specialty rankings, which we call specialty indicator. For instance, for a hospital whose specialty rankings are $[2, 0, 0, 2, 1, 1, 2, 0 \ldots, 2]$ (i.e. hospital 28 as in Table **??** in the Appendix), the specialty indicator is $\frac{1}{2} + \frac{1}{2} + \frac{1}{1} + \frac{1}{1} + \frac{1}{2} + \frac{1}{2} = 3$. The larger the specialty indicator is, the less specialized the hospital is. The study result in Figure 2 shows the inverse relationship between the specialty indicator and success rate. Specifically, when the value of specialty indicator is small (e.g., ≤ 1), the success rate

tends to be large.

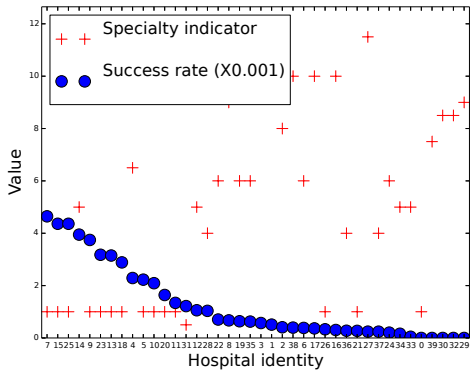


Figure 2: Hospital specialization

A.3 Multi-Threaded Precomputation

CPU Our CPU implementation of pre-computation is by using the pthread library in C++. Given our server is with 32 hardware threads, we spawned $64 = 2^6$ software threads to run the pre-computations. During the execution each software thread serially handles multiple possibilities.

GPGPU The GPGPU implementation is on a NVidia-Tesla GPU with specification illustrated in Table 2. Specifically, the GPGPU has global memory of 5 GB. All GPU threads are executed in 1 grid, which further consists $65,535$ blocks, each of which is of 1024 threads, resulting in 2^{27} threads in total. Our software implementation is based on CUDA library in C++. Given the pre-computation needs to compute multiple possibilities, we assign each possibility to one thread, thus handling about 27 parties. The key insight of using GPGPU is that different possibilities can be computed independently on different threads without any inter-thread synchronization.

A.4 Performance Measurement

To measure the performance, we mainly used four metrics, the number of AND gates in the compiled circuit, end-to-end execution time, memory consumption and communication costs.

- The GMW's Boolean circuit consists of XOR and AND gates. However, only AND gates are resource/time consuming because the XOR gates can be essentially evaluated locally for free (i.e. the free-XOR technique [1]). Thus, we only report the number of AND gates for performance study.
- We report the wall-clock time from the beginning of launching the first process to the end of the last process finishing its computation.
- We report the heap memory size which in GMW stores all circuit gates. This is measured by the Valgrind framework (particularly the Massif memory profiler [2]).
- We report the party-to-party communication overhead, by monitoring all outbound messages through the socket port of each process using IPTraf[1].

[1] http://iptraf.seul.org/

Table 2: Experiment platform

New York Server	
CPU	Xeon(R) E5-2640 v3 @ 2.60GHz 2 processors/16 cores/32 hyper-threads
Memory	245 GB
California Server	
CPU	Xeon(R) E5-2687W @ 3.10GHz 2 processors/16 cores/32 hyper-threads
Memory	256 GB
GPGPU	Nvidia Tesla K20c 1 grid/65535 blocks/2^{27} threads Global Memory 5119MB