Fractional FMD Scheme Implementation

Anonymized for CCS Rebuttal

1 Implementation

We implemented the FracFMD scheme in Go. We realized the uniformly ambiguous encryption scheme using an implementation of the H−ELG scheme with plaintext size $\ell = 128$ bits. We implemented the group $\mathbb{G}$ using the curve `secp256r1` provided in Go's `elliptic` library [Goo], using point compression for ciphertext serialization. To implement the sampling functions used in the garbled circuit, we wrote two short C programs computing $a \bmod B$ with hardcoded $B \in \{2^8, 2^{24}\}$ and a representing an integer of size 48 bits and 64 bits respectively (this encodes a $\kappa = 40$ bit statistical security parameter for the sampling function.) We compiled each circuit using the CBMC-GC compiler [FHK+14] (as packaged by Hastings *et al.* [HHNZ19].) Finally, we wrote our own garbled circuit implementation in Go using the BLAKE hash function for encryption, and employing both the point-and-permute and Free-XOR optimization of Kolesnikov and Schneider [KS08].

FracFMD denominator size (2^γ)	# non-XOR gates	Ciphertext (size)	KeyGen (time)	Flag (time)	Test (time)	Extract (time)
2^8	5,757 gates	318,530 bytes	0.298 ms	18.061 ms	9.585 ms	196 ns
2^{24}	21,925 gates	1,230,914 bytes	0.894 ms	66.672 ms	35.653 ms	355 ns

Table 1: Microbenchmarks for FracFMD. Times are computed using Go's benchmarking tool and using a random false positive rate. All experiments use the statistical parameter $\kappa = 40$. We give the complexity of the sampling circuit in terms of the number of non-XOR gates because XOR gates do not have any impact on the garbled circuit size.

Microbenchmarks for Fractional FMD We evaluated the FracFMD scheme with parameters $\gamma = 8$ and $\gamma = 24$ using statistical security parameter $\kappa = 40$ for both cases, assuming optimizations for Free-XOR and point-and-permute with a 120-bit encryption key size.[1] We observe that encryption and test times in this scheme depend on the circuit size, which is a function of the *denominator* and not the numerator a specific receiver selects. Hence we ran our benchmarks on random numerator values using a test script that follows the steps outlined in the previous paragraph. For a setting that supports fractional values

[1]We store labels in a 16-byte field, using 15 bytes to store labels and 1 byte to store the selector bit for point-and-permute (padded with 7 random bits.)

$x/2^8$ we calculated a ciphertext size of 318,530 bytes. For a setting that supports fractional values $x/2^{24}$ we calculated a ciphertext size of 1,230,914 bytes. Overall, our results indicate that the fractional FMD scheme has relatively small overhead in terms of encryption and test operations; the major cost (as expected) is the large ciphertext size, mainly due to the use of garbled circuits. This motivates the development of improved ambiguous functional encryption, which could produce more efficient fractional FMD constructions.

References

[FHK+14] Martin Franz, Andreas Holzer, Stefan Katzenbeisser, Christian Schallhart, and Helmut Veith. CBMC-GC: An ANSI C Compiler for Secure Two-Party Computations. In Albert Cohen, editor, *Compiler Construction - 23rd International Conference, CC 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, volume 8409 of *Lecture Notes in Computer Science*, pages 244–249. Springer, 2014.

[Goo] Google. Golang elliptic curve library.

[HHNZ19] M. Hastings, B. Hemenway, D. Noble, and S. Zdancewic. SoK: General purpose compilers for secure multi-party computation. In *2019 IEEE Symposium on Security and Privacy (SP)*, pages 1220–1237, 2019.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free xor gates and applications. In *International Colloquium on Automata, Languages, and Programming*, pages 486–498. Springer, 2008.