# Private Membership Test with Homomorphic Encryption

Sara Ramezanian, Tommi Meskanen, Valtteri Niemi
University of Helsinki
Helsinki, Finland
sara.ramezanian, tommi.meskanen, valtteri.niemi@helsinki.fi

***Abstract*—The general trend towards providing services from the cloud, raises the following question: "How to preserve end user privacy when querying cloud-hosted databases?"**

**Consider a realistic scenario where a client of an anti-malware company holds a file. This client wants to query the company's malware database to check whether the file is clean. This query is considered to be privacy-sensitive for the files that are supposed to be private, such as document files. One trivial approach to this problem is sending a copy of server's database to the client. However, due to the high bandwidth usage, this solution is infeasible in practice. Another solution is where the client sends a hash value of the file to the server and the server checks the file's hash value against the hash values of the malware samples. Due to the one-wayness property of the hash functions, it is not possible to derive contents of arbitrary files from their hash values. However, the server may store hash values of files that are of particular interest and is then able to check whether the client's file is among those.**

**The problem of running a set membership test in private manner is called private membership test (PMT). There are two parties involved in this setting: a client has an item and wants to check whether it is included in a set controlled by the server. Additionally, the client wants to have the option of not revealing the item, and the server does not want to reveal the contents of the whole set.**

**This study focuses on the PMT problem and presents three single-server protocols to resolve this problem. In our presented protocols, we use a probabilistic space-efficient data structure that is called Bloom filter, to store the server's database.**

**Protocol 1 utilizes Goldwasser-Micali homomorphic encryption to encrypt the Bloom filter t hat r epresents t he server's database. Protocol 2 uses RSA blind signature to encrypt the** database. We then store this encrypted database into a Bloom filter. Therefore, utilizing protocol 1 or 2 results to construction **of a privacy preserving variant of Bloom filter. In protocol 3, we divide the database into different subsets, insert each subset into a Bloom filter and finally arrange a matrix with the filters as elements of the matrix. We use Paillier homomorphic encryption to retrieve the value of one position in this matrix.**

**We have implemented our protocols in a realistic scenario where the server holds roughly two million malware fingerprints. In order to get realistic results, we measure the time complexity of the protocols by a commodity processor.**

**The three suggested solutions differ from each other, from privacy perspective and also from complexity point of view. Therefore, their use cases are different and it is impossible to choose one that is clearly the best between all three.**

**While Protocol 3 performs significantly faster in off-line phase than Protocols 1 and 2, its computation complexity is slightly higher than that of the other two protocols.**

**The implementation shows that the on-line phase of all the proposed protocols can be carried out in less than one second. However, the first two protocols are slightly faster than Protocol 3 in the on-line phase.**

**In the setting of Protocols 1 and 2, the client downloads a privacy-preserving variant of a Bloom filter. If the database requires a frequent updates, then the client may have to download the filter upon each query.**

**In conclusion, our implementation shows that the proposed protocols can be used in real world applications, for example, for Android app or website reputation services.**

ISSN 2305-7254