Privacy Preserving OpenPGP Public Key Distribution with Spamming Resistance*

Wenyuan Li[1,2], Wei Wang ✉[1,3], Jingqiang Lin[4], Qiongxiao Wang[1,2], and Wenjie Wang[1,2]

[1] State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing 100089, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing 100089, China
[3] Data Assurance and Communication Security Research Center, CAS, Beijing 10089, China
[4] School of Cyber Security, University of Science and Technology of China, Hefei 230027, Anhui, China
wangwei@iie.ac.cn

Abstract. OpenPGP public key distribution via Synchronizing Key-Servers (SKS) is facing the challenges of user privacy leakage caused by keyword search behaviors and service unavailability of OpenPGP software caused by OpenPGP certificate spamming attack (CVE-2019-13050). Most existing solutions to the problem dispense with the necessary features and functions of SKS or Web of Trust (WoT) for attack mitigation. In this paper, we put forward a solution which is privacy-preserving and spamming-resistant, while maintaining the functionalities of SKS and WoT. Considering the characteristics of our scenario, we protect user privacy by introducing a third-party server, and propose a specific third party-based private set intersection protocol to improve usability of OpenPGP software. Our protocol helps users filter out the required key data by intersection computation between *unbalanced* sets of keywords. We also propose an enhanced scheme for multi-key query and further privacy protection. We evaluate the usability and privacy of our schemes. Experimental results show that our scheme can largely reduce unnecessary data download with appropriate filter parameters. The proposed solution relies on the security of Elliptic Curve Diffie–Hellman, HMAC-based Key Derivation Function, Bloom filter and symmetric cryptographic encryption to defend against semi-honest adversaries.

Keywords: Pretty Good Privacy · Key Distribution · Keyword Search · Private Set Intersection.

* ✉Wei Wang is the corresponding author. This work was supported by National Key R&D Program of China (Grant No. 2020YFB1005800) and National Natural Science Foundation of China (Grant No. 61772518)

1 Introduction

Pretty Good Privacy (PGP) is an encryption program that provides cryptographic privacy and authentication for end-to-end data communication especially for e-mail communication. OpenPGP is a standard for e-mail encryption originally derived from PGP. PGP adopts Web of Trust (WoT) structure for distributed key management. Besides Public Key Infrastructure (PKI) with Certificate Authorities (CA), WoT is another mechanism for verifying digital identities on the Internet. In WoT, any entity can certify another entity by signing a certification signature for the binding relationship between a public key and the signee's identity. Certification signatures are the basis of WoT. Researchers have investigated the properties of WoT [37] which is similar to Online Social Network (OSN) such as Facebook and Twitter.

Since the beginning of year 2000, a wirte-only keyserver pool named Synchronizing KeyServers (SKS) has been employed to OpenPGP public key distribution. To date, SKS has provided over 6 million OpenPGP public keys[5] for users to query without restriction. Other major key distribution approachs include Web Key Directory (WKD) [24] and DNS-Based Authentication of Named Entities (DANE) [39]. WKD over HTTPS only provides keys without signatures. DANE simplifies key content by adding only a few signatures that are considered helpful. OpenPGP public key distribution via SKS has two disadvantages. One disadvantage is user privacy leakage caused by query behaviors. The other is low availability of OpenPGP software caused by OpenPGP certificate spamming attack.

On the one hand, the behaviors of users to query OpenPGP public keys from SKS will reveal the privacy of users. The attackers may infer the identity of users on the basis of the query information and the properties of WoT. Querying OpenPGP public keys via SKS can be regarded as a fundamental database operation called Keyword Search (KS) [15]. KS involves two parties: i) a server, holding a database comprised of a set of payloads and associated keywords. ii) a client, who may send queries consisting of keywords and receive the payloads associated with these keywords. Searchable encryption [4] is a positive way to protect sensitive data of users, which supports keyword search on encrypted data. However, the huge amount of legacy data which is already known to SKS in plain text renders the searchable encryption approach unrealistic.

On the other hand, the attackers of OpenPGP certificate spamming attack [1] expanded the size of OpenPGP public keys in SKS by spamming a quantity of signatures or large size signatures. The attack caused the failure of key import in OpenPGP software such as GNU Privacy Guard (GPG) and other usability issues. To mitigate the negative effects of the attack, GPG has updated a new version [23] to ignore all the signatures except self-signatures received from SKS. A draft [17] has proposed several simple mitigation methods mainly from the perspective of keyservers. Moreover, a new keyserver named keys.openpgp.org [2] separates an OpenPGP public key into identity informa-

[5] https://pool.sks-keyservers.net/pks/lookup?op=stats

tion and non-identity information. Identity information is only distributed with consent, and certification signatures in non-identity information are not distributed via keys.openpgp.org. Most of the mitigation recommendations and measures above suggest to obsolete SKS, eliminate the features of WoT, and put restrictions on user behaviors, which goes against the original intention of PGP developers.

For privacy protection, we first reconstruct the database in SKS into pairs of payloads and associated keywords, and split certification signatures and other key information by disparate keywords. Then we introduce a third-party server which helps users to search on encrypted pairs in SKS with encrypted keywords in requests. To mitigate OpenPGP public key distribution attack with the reservation of SKS and WoT, one solution is to give users the option to filter the OpenPGP public key data before downloading by computing intersection between sets of keywords in the user requests and in the database.

The research efforts of Private Set Intersection (PSI) focus on protocols [20, 7, 22] for the case where two parties holding sets compare encrypted versions of these sets in order to compute the intersection. While these protocols turn out to be impractical for use-cases like OpenPGP public key distribution with SKS. In our scenario, the PSI protocol needs a third-party server and should be suitable for *unbalanced* sets when keys queried have been spammed. We propose a third party-based PSI protocol specially for our scenario, and apply Elliptic Curve Diffie–Hellman (ECDH), HMAC-based Key Derivation Function (HKDF), Bloom filter, and symmetric cryptographic encryption to ensure security. The third-party server for PSI computation in our scheme is called PSI Proxy. In summary, we frame our key contributions as follows:

- We solve the problems of OpenPGP public key distribution with SKS on the basis of the original features and normal functions of SKS and WoT for key validation and verification.
- We first propose an OpenPGP public key distribution scheme for private preserving and spamming resistance with a third party-based PSI protocol, and an enhanced scheme with better privacy protection.
- We evaluate our schemes in terms of usability through user behavior simulations and privacy in the semi-honest model. Our scheme can largely reduce unnecessary data download, and has the high ability to resist various attacks.

The remainder of this paper is structured as follows. We review our research background in Section 2. In Section 3, we describe the problem in brief, and give our design principles and threat model afterwards. In Section 4, we describe our basic and enhanced schemes for OpenPGP public key distribution with a third party-based PSI protocol. We provide an evaluation of our scheme in terms of usability and privacy in Section 5. Finally, we summarize related work in Section 6, and give a conclusion in Section 7.

2 Background

To start, we focus on the status quo and properties of OpenPGP public key distribution, and introduce cryptographic building blocks that are required for the remainder of this work.

2.1 OpenPGP and Web of Trust

OpenPGP is a standard [13, 12] for providing end-to-end security for e-mail communication. In OpenPGP, User ID (including a user name and e-mail address) identifies a user, and is associated with a public/private key pair (either DSA/ElGamal or RSA) held by the user. Users can issue certification signatures to each other by signing the binding relationship between a public key and an identity with their private keys. These certification signatures are a significant part of a valid OpenPGP public key. An OpenPGP public key consists of a number of records called packets. OpenPGP packets are assembled into sequences in order to transfer keys. There is a public key ID in each Public-Key packet and an issuer key ID in each Signature packet. The essential elements of a transferable OpenPGP public key are as follows:

- One Public-Key packet (each contains a public key ID)
- Zero or more revocation signatures
- One or more User ID packets
- After each User ID packet, zero or more Signature packets (each contains an issuer key ID)
- Zero or more User Attribute packets
- After each User Attribute packet, zero or more Signature packets
- Zero or more Subkey packets
- After each Subkey packet, one Signature packet, plus optionally a revocation

In PGP environment [40], a key that is not revoked or expired is valid if it is users own key, or it is certified(signed) by other valid and trusted keys. Certification signatures are shared on keyservers, and show that the issuer is sure about the signee's identity. While trust is only locally (by users themselves) and not shared, and defines whose certification signatures users trust for validating other's keys. In GPG, the trust model is similar, but the trust information is specifically stored in a local file named *trustdb*. Users sign each other's keys and progressively build a web of public keys interconnected by these certification signatures which is so-called WoT [3].

2.2 Synchronizing Keyservers

SKS is widely used for OpenPGP public key distribution, and typically exists as a keyserver pool which consists of several keyservers. The main innovation of SKS is a highly-efficient set reconciliation algorithm [28] for keeping the keyservers synchronized. The algorithm reconciles two similar sets held by different hosts.

OpenPGP users communicate with SKS using OpenPGP HTTP Keyserver Protocol (HKP) [35]. Developers have primitively designed SKS to be write-only, so that the government can not forcibly delete or tamper the key data in SKS. The original design purposes of SKS and WoT are necessary for key validation and trust transitivity, which results in the difficulty to deal with OpenPGP certificate spamming attack. The reasons for the attack are as follows: i) OpenPGP does not restrict the maximal amount of signatures in an OpenPGP public key. ii) SKS only writes and does not validate the correctness and completeness of uploaded keys or signatures. Any user can verify a key in SKS and upload a certification signature to SKS without authentication. iii) GPG or other OpenPGP software has limitations on the size of the keys imported. The software may crash when the imported data blocks become too large. iv) SKS runs a reconciliation algorithm implemented as a software developed in Ocaml which is complex to maintain.

2.3 Private Set Intersection

PSI is a cryptographic technique of secure multiparty computation (MPC). It allows two parties holding sets to compare encrypted versions of these sets in order to compute the intersection. In traditional scenario, neither party reveals anything to the counterparty except for the elements in the intersection. While in the server-client scenario, only the client learns the intersection of her set with the set of the server, without the server learning intersection of his set with the clients [32]. A naive solution of PSI has been proposed that both parties apply a cryptographic hash function to their inputs and compare these hash values, which is efficient but insecure. The researchers have proposed a variety of advanced PSI protocols based on public key, oblivious transfer, circuit, third party and so on. Inbar et.al [20] implemented PSI based on secret sharing and Garbled Bloom Filter (GBF) as a variant of Bloom filter. Chen et al. [7] constructed a fast PSI protocol with a small amount of communication between two parties using full homomorphic encryption. This protocol is specially designed for the application scenario when one of the two sets is much smaller than the other. These two sets are named *unbalanced* sets. Third Party-Based PSI [22] realized a multi-party PSI protocol aided by a third-party server.

2.4 Bloom Filter

Bloom filter is a data structure that was conceived by Burton Howard Bloom in 1970 to retrieve whether an element is in a collection [5]. Bloom filter will tell either "possibly in set" or "definitely not in set", and the degree of false positive rate can be configured. Bloom filter represents a set X of n elements by an array of m bits, and uses k independent hash functions. We describe below the initial, add and check process of Bloom filter.

1) Before initialization, the generation parameters of Bloom filter m and k are calculated according to n and an intended false positive rate p.

$$m = -\frac{n \times \ln p}{(\ln 2)^2} \qquad k = \ln 2 \times \frac{m}{n} \tag{1}$$

2) During initialization, an array with the length of m bits is generated, and each bit is initialized to 0.
3) When adding an element s of X to Bloom filter, s is hashed with k hash functions to get k indices and set 1 to these indices of Bloom filter.
4) When checking whether an element s' belongs to X, s' is hashed with k hash functions to get k indices. If all these indices of Bloom filter are 1 then s' is considered to probably in X, otherwise s' is not in X.

2.5 HMAC-based Key Derivation Function

HKDF [25] is a special Key derivation function (KDF) [6] based on HMAC. HKDF function is divided into two phases: Extract and Expand. The Extract phase converts the input key into a short key that satisfies the pseudo-random nature. The Expand phase expands the pseudo-random key to the desired length. The inputs of HKDF function contain a hash function, a source key material, a extractor salt (which may be null or constant), a number of key bits to be produced by HKDF, and a "context information" string (which may be null) [26]. The number and length of the output key depend on the particular cryptographic algorithm that requires the key.

3 Assumptions and Goals

To present our proposal, we introduce the existing system of OpenPGP public key distribution with SKS in brief, and point out its drawbacks. Based on the issues in the existing system, we establish our design principles and threat model.

3.1 Problem Description

We provide the existing system of OpenPGP public key distribution via SKS in Fig.1. Users are able to connect to SKS by setting an access point such as pool.sks-keyservers.net in their OpenPGP software, and retrieve keys from SKS with *keyword* like Key ID, User ID or Fingerprint. After sending a query request to one of the keyservers, users will receive a response with *payload* (which is usually a complete OpenPGP public key) associated with *keyword* in OpenPGP format from the keyserver. Users can import *payload* into local keyring by using an OpenPGP software if needed. In the above-mentioned system, usability is lessened when users download or import *payload* with a quantity of signatures which have been spammed into keys. Upon most occasions, a tiny minority of these signatures are helpful for key validation or trust transitivity. At the same

time, user privacy is revealed when a user query SKS with *keyword*. All the OpenPGP public keys in SKS are retrievable to users with no authentication mechanisms. Moreover, the user communicates with SKS using HKP protocol, which is considered insecure. There is probably a relationship between the user and the owner of the key associated with *keyword*. The attackers may infer the user's identity on the basis of the query information and the properties of WoT.

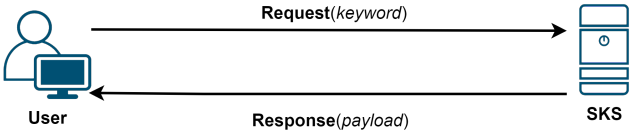


Fig. 1. Existing OpenPGP public key distribution system

3.2 Design Principles

Considering the privacy and usability issues of the existing OpenPGP public key distribution system, our scheme aims to achieve the following goals.

1) **Compatible**: Users can continue to use the original functions of SKS and WoT for key validation and trust transitivity. There is no modification required to the configuration of OpenPGP software for users to retrieve *payload* which contains certification signatures.
2) **Efficient**: Users can filter *payload* before downloading with specific filter conditions. The filtration can reduce user downloads of *payload* which further mitigates OpenPGP certificate spamming attack. Users can only import useful signatures for key validation and trust transitivity (For example, issuer key ID of a signature and Key ID of an introducer in *trustdb* are the same). We allow a moderate number of false positives to be introduced to hide the accurate number of filter conditions.
3) **Anonymous**: Attackers including semi-honest SKS and PSI Proxy can not obtain any private data in the query or filter requests of users, and thus are incapable of inferring the attributes of a user.

3.3 Threat Model

We develop our threat model with semi-honest adversaries including SKS and PSI Proxy. There is no collusion between SKS and PSI Proxy. An attacker seeks to learn the private information of a user. The private information consists of *keyword* of query requests, filter conditions (like *trustdb*) and identity information (like User ID). We list four types of possible attacks in the key distribution process. An attacker can first perform a dictionary attack, an eavesdropping attack or/and a replay attack to obtain the query or filter information, and then perform an attribute inference attack to infer user identity or other attributes. While the IP address tied to requests, a user can rely on a mix network such as Tor. IP address issue is out of scope of our threat model.

Dictionary Attack: In OpenPGP, the data type and length of *keyword* have been explicitly defined. Key ID is defined as a fixed eight-octet scalar. User ID is an UTF-8 text with no restrictions on its content. The fingerprint of a key is a MD5 hash which has been deprecated or a 160-bit SHA-1 hash. Given the format and length of *keyword*,a dictionary can be pre-computed by an attacker. The attacker can create requests with *keyword* in the dictionary, and attempt to collide with the query or filter information of any other users with these requests.

Eavesdropping or Replay Attack: In an eavesdropping attack, an attacker may take advantage of unsecured network communications to access the request and response of a query between users, SKS, and PSI Proxy. In a replay attack, an attacker can intercept a request of a user and re-transmit it to acquire *payload*. As all the *payload* in SKS and the local trust information of the user are constantly updated, the attacker can compare the present data to the previous ones, and may learn the private information of the user from the delta data.

Attribute Inference Attack: In attribute inference attack [19] towards OSN, an attacker aims to propagate attribute information of social network users with publicly visible attributes to users with missing or incomplete attribute data. Further, attackers can identify users in OSN with limited information [38]. In attribute inference attack towards WoT, an attacker may infer user identity on the strength of *keyword* and filter conditions. There may be a relationship between the user and the owner of the key queried or trusted. Based on the relationships and the characteristics of WoT, the user's attributes such as OpenPGP public key ID, User ID or its circle of friends can be inferred. Attackers can take advantage of these attributes to attack not only the user but also their friends through spam, XSS, phishing or malware which have occurred in OSN [14].

4 Scheme

Refer to our design principles, we propose an OpenPGP public key distribution system which introduces a third-party server PSI Proxy for PSI computation in Fig.2. When a user sends a query request with *keyword* to SKS, SKS sends processed *payload* associated with *keyword* to PSI Proxy. Then the user sends a filter request with filter conditions such as *trustdb* to PSI Proxy. PSI Proxy filters *payload* by running a specific PSI protocol between the sets of *keyword* in *trustdb* and *payload*, and returns filtered *payload* to the user.

Our design relies on a combination of ECDH[6], HKDF, Bloom filter[7] to address all the risks outlined in our threat model. Here, we explain the reason why the existing PSI protocols can not be directly used in our scenario. Then we depict our basic scheme with a PSI protocol dedicated to OpenPGP public

[6] Our scheme can use other key exchange algorithms such as Diffie-Hellman (DH).

[7] Bloom filter can be replaced by Cuckoo filter [11]. Both filters are very fast and compact, and may return false positives as answers to set-membership queries.

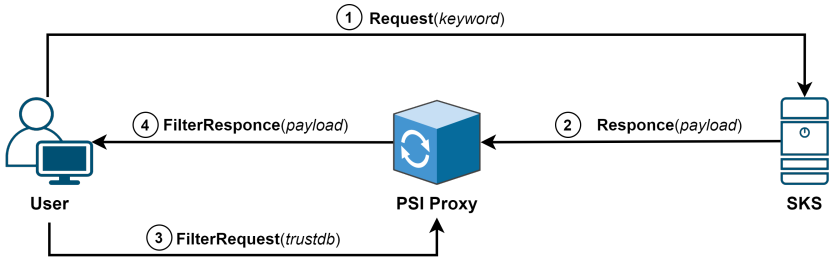


Fig. 2. Privacy Preserving and spamming resisting OpenPGP public key distribution system

key distribution, and detail the data exchanged between a user, SKS, and PSI Proxy. We also present a strengthened scheme for further privacy protection, and provide suggestions for optimization to improve compatibility and availability.

4.1 Protocol Selection

The PSI protocol in our scheme should meet two requirements. The first one is filtering *payload* to reduce downloads. The second one is hiding the query and filter information both in content and in number. We observe the characteristics of our scenario, and classify our scenario into one of PSI application scenarios. All the *payload* in SKS is openly searchable. Thus we need to select the PSI protocols that apply to the server-client scenario. In addition, when a key suffers from a certificate spamming attack, the number of signatures spammed into the key is much larger than the number of introducers in *trustdb*. Therefore, PSI protocols for *unbalanced* sets are proper for our scenario.

We focus on several existing PSI protocols which may fulfill our requirements. The PSI protocol using secret sharing and GBF requires users to receive complete data, which can not achieve the purpose of filtering. When applying the PSI protocol using full homomorphic encryption for *unbalanced* sets to our scenario, users still need to send the intersection results as *keyword* to get associated *payload* from SKS. SKS can learn the filter information from the intersection. The PSI protocols using a third party can hide the query information from SKS, but the third-party server simply computes the intersection and can not index *keyword* to the corresponding *payload*. Thus, we design a PSI protocol that is proposed specifically for OpenPGP public key distribution scenario.

4.2 Scheme Details

Our scheme consists of two main phases: Query phase and Filter phase. In Query phase, a user sends a query request with *keyword* to SKS. SKS processes the *payload* and associated *keyword*, and sends them to PSI Proxy. In Filter phase, the user sends *trustdb* in a filter request to PSI Proxy. PSI Proxy filters the processed *payload* by running a PSI protocol between *trustdb* and *keyword*. We

Symbol	Description
H	a public set that contains n hash functions, $H = \{h_1, h_2, h_3, \ldots, h_n\}$
K	a symmetric key generated by ECDH
IV	an initialization vector for symmetric encryption
qid	an auto-increment ID in a long long int type for each query
L	a number of key bits produced by HKDF
XTS	a secret extractor salt of HKDF
$CTXinfo$	a "context information" string of HKDF
err_rate	an intended false positive rate of Bloom filter
n_h	a number of hash functions of Bloom filter
m	a length of Bloom filter
pid_i	public key ID of ith key
$sid_{i,j}$	issuer key ID of jth signature in i^{th} key
tid_i	Key ID of ith introducer in $trustdb$
$info_i$	Other key information except signatures in i^{th} key
$sig_{i,j}$	j^{th} signature in i^{th} key
$info_pair_i$	the pair of $info_i$ and associated pid_i, $info_pair_i = (pid_i, info_i)$
$sig_pair_{i,j}$	the pair of $sig_{i,j}$ and associated Key IDs (including pid_i and $sid_{i,j}$), $sig_pair_{i,j} = ((pid_i, sid_{i,j}), sig_{i,j})$
S_1	the database of $info_pair_i$ in SKS
S_2	the database of $sig_pair_{i,j}$ in SKS
S	the entire database in SKS, $S = S_1 \cup S_2$

Table 1. Notation list

define the variants in our scheme, and list them in Tab.1. We reconstruct the database S in SKS as two kinds of pairs: $info_pair_i$ represented as $(pid_i, info_i)$, and $sig_pair_{i,j}$ represented as $((pid_i, sid_{i,j}), sig_{i,j})$. Before querying, the user and SKS generate K, and then safely transmit other parameters including IV, qid, n_h ($n_h \leq n$), L and XTS with K.

Query Phase: When querying an OpenPGP public key $\hat{pk}$, an user sends public key ID $\hat{pid}$ of $\hat{pk}$ as $keyword$[8]. SKS finds the pairs (both $info_pair_i$ and $sig_pair_{i,j}$) having $\hat{pid}$ as $keyword$. For these pairs, SKS calculates HKDF values $SKD_{i,j}$ of issuer key IDs, and encrypts all the $payload$ using Algorithm 1 to complete the data processing in Eq.(2). SKS uses $sid_{i,j}$ as SKM and qid as $CTXinfo$ for HKDF function. After processing, SKS sends qid, $enc_{\hat{i}}$, and the set of processed pairs $Sig_pair_{\hat{i}}$ to PSI Proxy.

$$((pid_i, sid_{i,j}), sig_{i.j}) \rightarrow ((pid_i, SKD_{i,j}), Enc_K\,(sig_{i,j})) \tag{2}$$

Filter Phase: The user locally adds tid_i to Bloom filter BF_c by HKDF values TKD_i of tid_i in $trustdb$, which is outlined in Algorithm 2. The user uses tid_i

[8] We choose Key ID as $keyword$ to make our solution more concise. Users can also use User ID or fingerprint in the implementation.

Algorithm 1 ProcessData

Input: $S_1, S_2, IV, H, n_h, XTS, qid, L$
Output: $enc_{\hat{i}}$, $Sig_pair_{\hat{i}}$

```
for (pid_i, info_i) ∈ S_1 do
    if pid_i = p̂id then
        î ← i
        enc_î ← Enc_K (info_î)
        for j = 1; j ≤ n_{s_î}; j++ do
            for k = 1; k ≤ n_h; k++ do
                SKD_{î,j}[k] ← HKDF(h_k, XTS, sid_{î,j}, qid, L)
            end for
            enc_{î,j} ← Enc_K (sig_{î,j})
            Sig_pair_î[j] ← ((pid_î, SKD_{î,j}), enc_{î,j})
        end for
    end if
end for
```

as SKM and qid as $CTXinfo$ for HKDF function. Then the user sends qid, BF_c, and n_h in a filter request to PSI Proxy. m and n_h of BF_c are calculated by err_rate and the set size n_t of $trustdb$ using Eq.(1) in 2.4. PSI Proxy searches $Sig_pair_{\hat{i}}$ according to qid as the input of Algorithm 3. The output Sig which contains n_{sf} signatures is sent to the user together with $enc_{\hat{i}}$ as $payload$. All the data received by the user can be decrypted with K and IV, and imported into local keyring.

Algorithm 2 CreateFilterRequest

Input: $err_rate, n_t, H, n_h, XTS, trustdb, qid, L$
Output: BF_c

```
initial(BF_c, err_rate, n_t)
for tid_i ∈ trustdb do
    for k = 1; k ≤ n_h; k++ do
        TKD_i[k] ← HKDF(h_k, XTS, tid_i, qid, L)
        add(BF_c, TKD_i[k])
    end for
end for
```

4.3 Extension to Public Key IDs

Our scheme applies for single-key queries, and can not hide public key IDs queried from SKS. We suggest users to rely on Tor to query. On the basis of our scheme, we propose an enhanced scheme shown in Fig.3 by applying a "double filtration" solution. As the number of keys queried is far less than the amount of keys in SKS, the PSI computation for issuer key IDs can also be applied to public key IDs. Especially when the user queries multiple keys in a query, this extension can further enhance privacy protection with a few downloads of *payload*.

Algorithm 3 ComputeSetIntersection

Input: BF_c, $Sig_pair_{\hat{i}}$
Output: Sig, n_{sf}

```
for j = 1, n_sf = 1; j ≤ n_{s_i}; j++ do
    iscontian ← check(BF_c, SKD_{î,j})
    if iscontain == 1 then
        ĵ ← j
        Sig[n_sf++] ← Enc_k(Sig_{î,ĵ})
    end if
end for
```

In Query phase, SKS needs to shuffle the orders of pairs before processing (in case semi-honest PSI Proxy acts as a user to get the indices of all the data). Then as shown in Eq.(3) and Eq.(4), SKS computes HKDF values PKD_i of pid_i, and processes all the pairs in S. SKS sends PKD_i and the processed database to PSI Proxy.

$$(pid_i, info_i) \rightarrow (PKD_i, Enc_K(info_i)) \tag{3}$$

$$((pid_i, sid_{i,j}), sig_{i.j}) \rightarrow ((PKD_i, SKD_{i,j}), Enc_K(sig_{i,j})) \tag{4}$$

In Filter phase, when querying n_q keys with a set of public key IDs named *qlist*, the user uses Algorithm 2 to add these keys into Bloom filter BF_q by HKDF values PKD_i (*err_rate* of BF_q can differ from that of BF_c). The user then sends BF_q to PSI Proxy. PSI Proxy uses Algorithm 3 to filter $info_pair_i$ in S_1 with BF_q as the first level of filtration, finding n_{kf} public key IDs that may be involved in *qlist*. PSI Proxy then performs Filter phase of the basic scheme towards $sig_pair_{i,j}$ having the same *keyword* pid_i in S_2 with BF_c as the second level of filtration, and sends the output as *payload* to the user. The user encrypts the filter result with K and IV, and selects the desired *payload* by *qlist*. While the limitation of this solution is for each query, SKS is required to process the entire database S, including HKDF calculation of all the *keyword* and symmetric encryption of all the *payloads*.

4.4 Optimization

On top of our scheme, several optimizations can be carried out to improve compatibility and availability. With no need to modify the original functions of SKS, one or more proxy servers can be deployed on SKS side. The proxy server takes charge of SKS for the interactions with users or PSI Proxy and *payload* processing to improve compatibility. Simplifying HKDF function to simple Hash function in our scheme helps decrease computation of SKS, which may suffer from a dictionary attack. However, the employ of Bloom filter still introduces false positives to hide the set size and content of *keyword*. By adjusting the parameter *err_rate* and L (not less than the hash function output length), the communication amount of HKDF values from SKS to PSI Proxy can be effectively reduced.

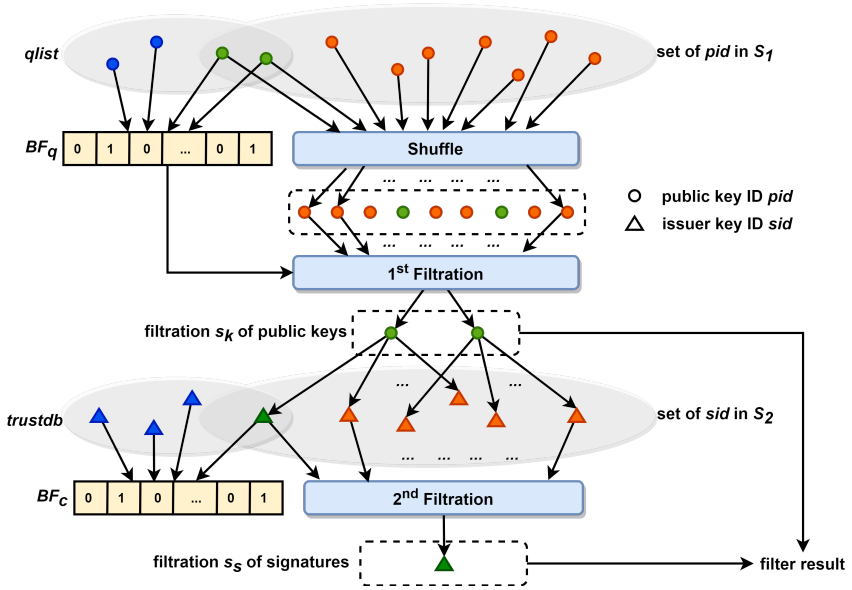


Fig. 3. An extended scheme to enhance privacy protection which applies two levels of filtration. Before filtering, SKS shuffles all the OpenPGP public keys in the database, and users generate BF_q with *qlist* and BF_c with *trustdb*. In first filtration, PSI Proxy filters *pid* with BF_q to get the intersection s_k of public keys. In second filtration, PSI Proxy filters *sid* which has certified *sid* in s_k with BF_c to get the intersection s_s of signatures. With all the Key IDs in s_k and s_s as *keyword*, PSI Proxy sends encrypted *payload* as filter result to users.

5 Evaluation

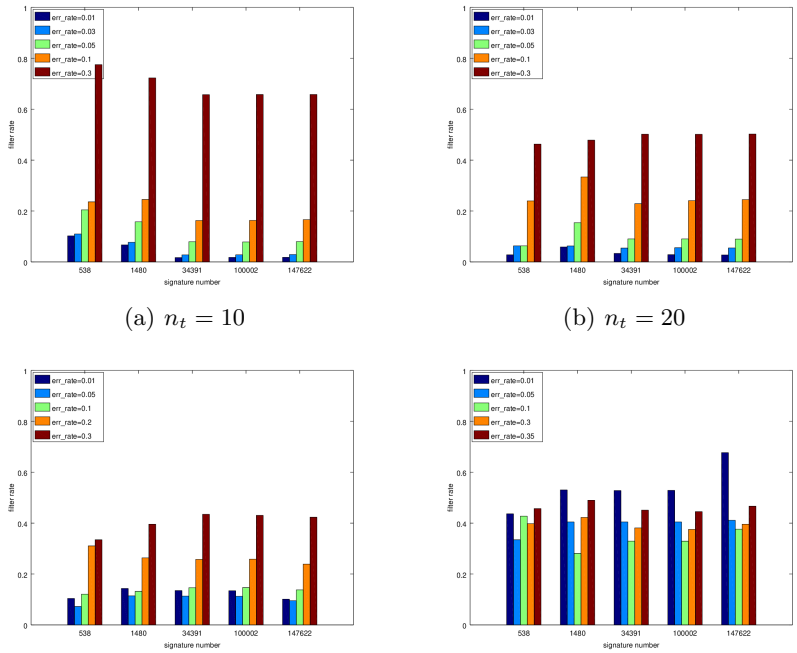
We evaluate our solutions in terms of usability and privacy. We set disparate benchmarks to evaluate the effect of filtering in the basic scheme. We also analyze the complexities and the storage required for the parties in both solutions. For the possible attacks in threat model, we evaluate the cryptographic operations and techniques in our scheme for the processing and transmission of sensitive data.

5.1 Experimental Settings

We implement our scheme including a GPG user, PSI Proxy, and one keyserver of SKS in Ubuntu 16.04 operating system and C++ programming language. We use OpenSSL library (v. 1.1.1) for 128-bit ECDH key exchange, HKDF computation, and AES-128-CBC symmetric encryption. We need to simulate the private information *qlist* and *trustdb* of users. As the characteristics of WoT are similar to those of OSN, With reference [29] to the number of friends in OSN, the number of key IDs in the user's local *trustdb* is assumed to be in the range of 0

to 100. We randomly generate Key IDs in *qlist* and *trustdb* in the normal range. We choose the main keyserver of SKS of which the URL is https://pool.sks-keyservers.net/. For an OpenPGP public key in SKS, the reasonable number of signatures should be less than or around 1000 [34] which can be regarded as a criterion for whether the key has been attacked. We find several representative keys that have suffered from OpenPGP certificate spamming attack in SKS. The user name of these keys are Yegor Timoshenko (174622 signatures), Robert J. Hansen (149120 signatures), Ryan McGinnis (100002 signatures), Patrick Brunschwig (151491 signatures), Lance Cottrell (34391 signatures).

Our scheme also consider data updates in SKS and in users' *trustdb*. There are constant updates on the key data in SKS such as revocations of the existing keys or signatures or the uploading of fresh keys or signatures. In the implementation, users send a query request with a parameter of time range, and receive the key data with updates in this time range from SKS. Users can modify the trust information in *trustdb* subjectively. When *trustdb* changes, the parameters of Bloom filter need to be modified by users to generate a new filter.



(a) $n_t = 10$ (b) $n_t = 20$

(c) $n_t = 50$ (d) $n_t = 100$

Fig. 4. Experimental results of filtration. We choose the set size n_t of *trustdb* $\in \{10, 20, 50, 100\}$ and the average set size of signatures of the target keys $n_s \in \{538, 1480, 34391, 100002, 147622\}$. We get the ratio *filter rate* of user downloads to the original key size under different *err_rate*.

5.2 Usability Evaluation

We compare the filtering effect in the basic scheme under diverse set sizes of *keyword* through the experimental results. As shown in Fig.4, the greater the disparity of the set sizes, the greater the mitigation effect of OpenPGP certificate spamming attack. Then we set different *err_rate* of Bloom filter to evaluate the impact of it on downloads. In most cases in Fig.4, as *err_rate* reduces, the decrease of downloads becomes more pronounced. We suggest *err_rate* to be less than 0.3 whthin which the reduction of downloads can be more than half. Supposing the average size of signatures is s, regardless of the fact that the size of signatures can be exaggerated, we get the decline d of downloads which can be approximately calculated as $n_s(1-p)s$ theoretically if n_s is large enough.

For the basic solution in 4.2 and the improved solution in 4.3, we depict the computation and communication complexities of users and SKS, and the storage required by PSI Proxy during key distribution in Tab.2. The complexities and storage are linear to the number of users. The computation complexity is expressed as the number of symmetric cryptographic operations (sym) and the number of HKDF operations (hkdf). ECDH operations which perform only once before each query are not represented in Tab.2. The communication complexity of a user in our schemes is much lower than that in the existing system for the most part, and the computation complexity depends mainly on the number of symmetric encryption operations. The complexities of SKS depend on the number of signatures in the keys queried in the basic scheme. While they depend on the total amount of pairs in SKS in the improved scheme. In addition, the data volume that PSI Proxy needs to store is approximately equal to the communication amount transmitted by SKS.

Role	**Type**	**Basic scheme**	**Improvement scheme**
User	Computation (#ops hkdf/sym)	$n_t n_h$ hkdf $n_{sf}+1$ sym	$(n_t+n_q)n_h$ hkdf $(n_{sk}+1)n_{kf}$ sym
	Communication (bit)	$\sigma+m+\ell+k+n_{sf}s$	$\sigma+2m+\ell+n_{kf}k+n_{kf}n_{sf}s$
SKS	Computation (#ops hkdf/sym)	$n_s n_h$ hkdf n_s+1 sym	$n_s n_k n_h$ hkdf $(n_s+1)n_k$ sym
	Communication (bit)	$\sigma+\ell+k+n_s s+n_s n_h L$	$\sigma+\ell+n_k k+n_k n_s s+n_s n_k n_h L$
PSI Proxy	Storage (bit)	$m+\ell+k+n_s s+n_s n_h L$	$2m+\ell+n_k k+n_k n_s s+n_s n_k n_h L$

Table 2. Complexities and storage for OpenPGP public key distribution (σ: bit size of the communication required for key exchange and data transmission before query; ℓ: bit size of *qid*; hkdf: HKDF operations; sym: symmetric cryptographic operations; n_k, n_{kf}: set sizes as defined in 4.2; n_s, n_{sf}: the average number of the signatures in each key and in the intersection; k, s: bit size of $info_i$ and $sig_{i,j}$ defined in 4.2; L: HKDF parameter; m, n_h: Bloom filter parameters).

5.3 Privacy Evaluation

Before the privacy evaluation of our proposal, we explain the security of cryptographic algorithms and techniques used in our solution which consists of ECDH, HKDF, Bloom filter and symmetric encryption. ECDH is one of key exchange schemes based on ECC which provides the same cryptographic strength as the RSA system, but with much smaller keys. We apply HKDF function on the basis of the properties of the HMAC scheme both as extractor and pseudorandom function. In our solution, we further strengthen the security of HKDF with a secret salt. For Bloom filter, a query returns either "possibly in set" or "definitely not in set" rather than an exact element in the set. The apply of Bloom filter also automatically introduces false positives for filtering, which can hide the accurate set size and content. For each query, a key for symmetric encryption is pre-shared between users and SKS, and is invalidated each time the user decrypts the result. In both our basic and extended scheme, all the sensitive data and parameters between users, SKS, and PSI Proxy are transmitted in a non-plaintext form. We carry out security analysis of our scheme towards possible attacks outlined in the threat model on the basis of the security of cryptographic algorithms and techniques.

Collision Resistant: The parameters of HKDF function are agreed between users and SKS in a secure channel. Without the parameters of other users, an attacker can not launch a dictionary attack by the computation of HKDF values for all the keywords. Although SKS can obtain the parameters and compute HKDF values, it can not create filter requests without *qlist* and *trustdb* of users. In our enhanced solution, we ask SKS to shuffle all the *payload* before processing to prevent semi-honest PSI Proxy. If SKS processes the *payload* in a fixed sequence, PSI Proxy can act as a user to query with all the keywords in the dictionary. Thus, PSI Proxy can learn the real data content by the sequential index. When a user filers *payload* through PSI Proxy, PSI Proxy can learn the privacy information of the user by the indices of filter results. After SKS shuffling all the keys, PSI Proxy can not learn the plaintext of *payload* by indices.

Eavesdropping and Replay Resistant: The attacker eavesdropping on users, SKS, and PSI Proxy can not learn any privacy information from the requests and responses which are all encrypted. Our schemes can also resist replay attacks by adding a query ID for each query. Query ID is auto-increment for each query and is associated with the symmetric key exchanged between users and SKS. The attacker can be detected by SKS or PSI Proxy when re-transmitting the query or filter requests with the same or uncorrect query ID.

Identity Anonymous: An attacker can not identify a user with no access to user privacy information particularly the filter information like *trustdb*. In the basic scheme, semi-honest SKS can only obtain *keyword* of the key queried. While SKS can not get either *qlist* as *keyword* and *trustdb* in the enhanced

scheme. Other attackers like semi-honest PSI Proxy can not obtain all the sensitive information. Both *qlist* and *trustdb* are in the form of Bloom filter, and *payload* processed by SKS is in the form of ciphertext. Moreover, the parameters of Bloom filter are constantly modified by users to generate new filters, so that attackers are agnostic about the variation of *qlist* and *trustdb*.

6 Related Work

PSI protocols for *unbalanced* sets have been used to protect user privacy in many application scenarios. Thomas et al. [36] introducing a PSI protocol based on ECDH key Exchange which is used to mitigate credential stuffing attack. Kales et al. [21] implemented a PSI protocol based on Oblivious Pseudo-Random Function (OPRF) and Cuckoo filter to protect the privacy of contact information of mobile devices. Related to PSI, private keyword search which is also introduced as Oblivious Keyword Search (OKS) [31] has become another problem to solve for privacy protection of query. Kushilevitz and Ostrovsky [27] first suggests a single-server Private Information Retrieval (PIR) protocol for obtaining a semi-private Keyword Search (KS) protocol. PIR schemes allow a user to retrieve the ith bit of an n-bit database, without revealing to the database the value of i. There are two classes of PIR protocols. Information-theoric PIR (IT-PIR) [9] provides security guarantees, and is usually more computationally efficient. However, any non-trivial IT-PIR requires multiple non-colluding servers. Chor et al. [8] proved that the trivial protocol in which clients are sent the entire database is communication optimal in the single-server setting. Computational PIR (cPIR) [10] can achieve sublinear communication with a single server, but is typically more computationally expensive as it usually involves cryptographic operations based on public-key primitives to be carried out on each element of the database. Other techniques for private keyword search are homomoephic encryption [16], oblivious transfer [33], oblivious-RAM (ORAM) [18], oblivious polynomial Evaluation [30] and searchable encryption [4].

7 Conclusion

In this paper, We first propose an OpenPGP public key distribution scheme with an effective third party-based PSI protocol for *unbalanced* sets to deal with user privacy leakage and certificate spamming attack. We enhance our scheme by applying a "double filtration" solution for further user privacy protection. Our schemes applis cryptographic operations and techniques to prevent privacy disclosure to defend against semi-honest adversaries. With appropriate filter parameters, our shemes can effectively reduce unnecessary user download to resist certificate spamming attack.

However, in our enhanced scheme, each query requires SKS to fully encrypt the database, which increases the computational and communication complexity. Future research may consider to combine PIR to remove the third party and perfect our proposal which can be implemented in a multi-user scenario.

References

1. CVE-2019-13050. https://nvd.nist.gov/vuln/detail/CVE-2019-13050 (2019)
2. keys.openpgp.org. https://keys.openpgp.org (2019)
3. Abdul-Rahman, A.: The pgp trust model. In: EDI-Forum: the Journal of Electronic Commerce. vol. 10, pp. 27–31 (1997)
4. Bellare, M., Boldyreva, A., O'Neill, A.: Deterministic and efficiently searchable encryption. In: Annual International Cryptology Conference. pp. 535–552. Springer (2007)
5. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM **13**(7), 422–426 (1970)
6. Camenisch, J., Fischer-Hübner, S., Rannenberg, K.: Privacy and identity management for life. Springer Science & Business Media (2011)
7. Chen, H., Laine, K., Rindal, P.: Fast private set intersection from homomorphic encryption. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1243–1255 (2017)
8. Chor, B., Gilboa, N., Naor, M.: Private information retrieval by keywords. Citeseer (1997)
9. Demmler, D., Herzberg, A., Schneider, T.: Raid-pir: Practical multi-server pir. In: Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security. pp. 45–56 (2014)
10. Dong, C., Chen, L.: A fast single server private information retrieval protocol with low communication cost. In: European Symposium on Research in Computer Security. pp. 380–399. Springer (2014)
11. Fan, B., Andersen, D.G., Kaminsky, M., Mitzenmacher, M.D.: Cuckoo filter: Practically better than bloom. In: Proceedings of the 10th ACM International on Conference on Emerging Networking Experiments and Technologies. p. 75–88. CoNEXT '14, Association for Computing Machinery, New York, NY, USA (2014). https://doi.org/10.1145/2674005.2674994, https://doi.org/10.1145/2674005.2674994
12. Finney, H., Donnerhacke, L., Callas, J., Thayer, R.L., Shaw, D.: OpenPGP Message Format. RFC 4880 (Nov 2007). https://doi.org/10.17487/RFC4880, https://rfc-editor.org/rfc/rfc4880.txt
13. Finney, H., Thayer, R.L., Donnerhacke, L., Callas, J.: OpenPGP Message Format. RFC 2440 (Nov 1998). https://doi.org/10.17487/RFC2440, https://rfc-editor.org/rfc/rfc2440.txt
14. Fire, M., Goldschmidt, R., Elovici, Y.: Online social networks: threats and solutions. IEEE Communications Surveys & Tutorials **16**(4), 2019–2036 (2014)
15. Freedman, M.J., Ishai, Y., Pinkas, B., Reingold, O.: Keyword search and oblivious pseudorandom functions. In: Theory of Cryptography Conference. pp. 303–324. Springer (2005)
16. Gentry, C., et al.: A fully homomorphic encryption scheme, vol. 20. Stanford university Stanford (2009)
17. Gillmor, D.K.: Abuse-Resistant OpenPGP Keystores. Internet-Draft draft-dkg-openpgp-abuse-resistant-keystore-04, Internet Engineering Task Force (Aug 2019), https://datatracker.ietf.org/doc/html/draft-dkg-openpgp-abuse-resistant-keystore-04, work in Progress
18. Goldreich, O.: Towards a theory of software protection and simulation by oblivious rams. In: Proceedings of the nineteenth annual ACM symposium on Theory of computing. pp. 182–194 (1987)

19. Gong, N.Z., Liu, B.: Attribute inference attacks in online social networks. ACM Transactions on Privacy and Security (TOPS) **21**(1), 1–30 (2018)
20. Inbar, R., Omri, E., Pinkas, B.: Efficient scalable multiparty private set-intersection via garbled bloom filters. In: International Conference on Security and Cryptography for Networks. pp. 235–252. Springer (2018)
21. Kales, D., Rechberger, C., Schneider, T., Senker, M., Weinert, C.: Mobile private contact discovery at scale. In: 28th {USENIX} Security Symposium ({USENIX} Security 19). pp. 1447–1464 (2019)
22. Kamara, S., Mohassel, P., Raykova, M., Sadeghian, S.: Scaling private set intersection to billion-element sets. In: International Conference on Financial Cryptography and Data Security. pp. 195–215. Springer (2014)
23. Koch, W.: Gnupg 2.2.17 released to mitigate attacks on keyservers. `https://lists.gnupg.org/pipermail/gnupg-announce/2019q3/000439.html` (2019)
24. Koch, W.: OpenPGP Web Key Directory. Internet-Draft draft-koch-openpgp-webkey-service-11, Internet Engineering Task Force (Nov 2020), `https://datatracker.ietf.org/doc/html/draft-koch-openpgp-webkey-service-11`, work in Progress
25. Krawczyk, D.H., Eronen, P.: HMAC-based Extract-and-Expand Key Derivation Function (HKDF). RFC 5869 (May 2010). https://doi.org/10.17487/RFC5869, `https://rfc-editor.org/rfc/rfc5869.txt`
26. Krawczyk, H.: Cryptographic extraction and key derivation: The hkdf scheme. In: Annual Cryptology Conference (2010)
27. Kushilevitz, E., Ostrovsky, R.: Replication is not needed: Single database, computationally-private information retrieval. In: Proceedings 38th annual symposium on foundations of computer science. pp. 364–373. IEEE (1997)
28. Minsky, Y., Trachtenberg, A.: Practical set reconciliation. In: 40th Annual Allerton Conference on Communication, Control, and Computing. vol. 248. Citeseer (2002)
29. Mislove, A., Marcon, M., Gummadi, K.P., Druschel, P., Bhattacharjee, B.: Measurement and analysis of online social networks. In: Proceedings of the 7th ACM SIGCOMM Conference on Internet Measurement. p. 29–42. IMC '07, Association for Computing Machinery, New York, NY, USA (2007). https://doi.org/10.1145/1298306.1298311, `https://doi.org/10.1145/1298306.1298311`
30. Naor, M., Pinkas, B.: Oblivious polynomial evaluation. SIAM Journal on Computing **35**(5), 1254–1281 (2006)
31. Ogata, W., Kurosawa, K.: Oblivious keyword search. Journal of complexity **20**(2-3), 356–371 (2004)
32. Pinkas, B., Schneider, T., Segev, G., Zohner, M.: Phasing: Private set intersection using permutation-based hashing. Cryptology ePrint Archive, Report 2015/634 (2015), `https://eprint.iacr.org/2015/634`
33. Rabin, M.O.: How to exchange secrets with oblivious transfer. IACR Cryptol. ePrint Arch. **2005**(187) (2005)
34. Richters, O., Peixoto, T.P.: Trust transitivity in social networks. PloS one **6**(4), e18384 (2011)
35. Shaw, D.: The OpenPGP HTTP Keyserver Protocol (HKP). Internet-Draft draft-shaw-openpgp-hkp-00, Internet Engineering Task Force (Mar 2003), `https://datatracker.ietf.org/doc/html/draft-shaw-openpgp-hkp-00`, work in Progress
36. Thomas, K., Pullman, J., Yeo, K., Raghunathan, A., Kelley, P.G., Invernizzi, L., Benko, B., Pietraszek, T., Patel, S., Boneh, D., et al.: Protecting accounts from credential stuffing with password breach alerting. In: 28th {USENIX} Security Symposium ({USENIX} Security 19). pp. 1556–1571 (2019)

37. Ulrich, A., Holz, R., Hauck, P., Carle, G.: Investigating the openpgp web of trust. In: European Symposium on Research in Computer Security. pp. 489–507. Springer (2011)
38. Vesdapunt, N., Garcia-Molina, H.: Identifying users in social networks with limited information. In: 2015 IEEE 31st International Conference on Data Engineering. pp. 627–638 (2015). https://doi.org/10.1109/ICDE.2015.7113320
39. Wouters, P.: DNS-Based Authentication of Named Entities (DANE) Bindings for OpenPGP. RFC 7929 (Aug 2016). https://doi.org/10.17487/RFC7929, `https://rfc-editor.org/rfc/rfc7929.txt`
40. Zimmermann, P.R.: The official PGP user's guide. MIT press (1995)