Polymorphic Encryption and Pseudonymisation of IP Network Flows

1st Abraham Westerbaan
Digital Security
Radboud University
Nijmegen, The Netherlands
bram@westerbaan.name

2nd Luuk Hendriks
Design and Analysis of Communication Systems
University of Twente
Enschede, The Netherlands
luuk.hendriks@utwente.nl

***Abstract*—We describe a system, *PEP3*, for storage and retrieval of IP flow information in which the IP addresses are replaced by pseudonyms. Every eligible party gets its own set of pseudonyms. A single entity, the *transcryptor*, that is composed of five independent peers, is responsible for the generation of, depseudonymisation of, and translation between different sets of pseudonyms. These operations can be performed by any three of the five peers, preventing a single point of trust or failure. Using homomorphic aspects of ElGamal encryption the peers perform their operations on encrypted –and potentially– pseudonymised IP addresses only, thereby never learning the (pseudonymised) IP addresses handled by the parties. Moreover, using Schnorr type proofs, the behaviour of the peers can be verified, without revealing the (pseudonymised) IP addresses either. Hence the peers are central, but need not be fully trusted. The design of our system, while easily modified to other settings, is tuned to the sheer volume of data presented by IP flow information.**

***Index Terms*—Network Flows, Polymorphic Encryption and Pseudonymisation.**

I. Introduction

The concept of network flow monitoring is well known and widely deployed by network operators. The payload of the packets, and thus the contents of the communication itself, is discarded in this aggregation process: a *flow* usually only contains the number of packets and bytes that were observed, the start and end time of that flow, and the 5-tuple[1] it was aggregated on.

Most routers commonly deployed by network operators can perform this flow aggregation, and export the flow statistics (often using either Cisco's NetFlow [1], or IPFIX [2], an IETF standard) towards a collector. The collector then stores, with a certain retention time, the received flow records either on disk or in a database, which allows the operator to query the data.

Even without the actual payload, flow measurements contain sufficient information for a plethora of use cases, such as keeping traffic statistics for network operators, large-scale measurement studies for researchers, or forensic activities by security teams. On the flip side flow information may reveal very sensitive information, despite not containing the actual contents of the communication. Consider for example the IP address of a server hosting only one website, about a rare disease. Whence we propose to reduce risk without sacrificing too much usability by replacing the IP addresses in the flow data by pseudonyms, and, moreover, to use a unique set of pseudonyms for every eligible *party* (storage facility, researcher, investigator, etc., further explained in Section II-B). This might be achieved by (deterministically and symmetrically) encrypting IP addresses destined for a certain party by a secret key unique to—but not shared with—that party. We hold that the *transcryptor*, the entity that keeps these secret keys, and is thus responsible for generating and translating the pseudonyms,

1) should not learn the IP addresses it processes;
2) should not have a single point of failure; and
3) should be verifiable.

In this paper we describe *PEP3*, a system that fulfils these requirements by:[2]

1) having the transcryptor deal with encrypted pseudonyms only, and leveraging homomorphic aspects of ElGamal[4] encryption, noted earlier in [3] (and arguably [5], [6]), to perform the required operations on these encrypted pseudonyms;
2) breaking the central secret keys held by the transcryptor into ten shares each, and dividing those shares over five independent *peers*—which together form the transcryptor—in such a way that every triple of peers, but no pair, together has all ten shares; and
3) verifying the honesty of the peers by (occasionally retroactively) requiring a Schnorr type proof (see Section III-B) for the performed operation.

We envision that an organisation wishing to run PEP3 would collaborate with multiple (up to five) independent and possibly geographically diverse hosting providers to run the peers, and would arrange by contract that the shares are not disclosed, even to the network operator itself.

In the design of any complex system there are a lot of knobs to turn, and dials to watch. We do not pretend that our system is optimal, nor that we have defined what optimal should mean in this context. We have, however, preferred simplicity and

This work is part of the research programme VWData with project number 400.17.605, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO).

[1]Source and destination addresses and ports, and protocol type.

[2]The name "PEP3" is a contraction of "PEP" the precursor to this system, [3], and "P3", our project's designation within the VWData programme.

speed over cleverness and additional features. Where possible, we have chosen tried and trusted cryptographic systems (curve25519, ElGamal encryption, Schnorr type proofs) over exciting new techniques (such as pairing based cryptography).

a) Contributions: PEP3 builds on the PEP system[3] designed to store medical research data. Our system differs in several ways from PEP, most notably in that we divide the three different roles (transcryptor, access manager, key server) in PEP over five peers. The general idea of splitting a global secret into shares, and dividing it over several peers, so that only a subset of them is needed to perform the action, is well-known[7] and fundamental to fields such as threshold cryptography and secure multiparty computation, as is the use of Schnorr-type proofs in this context. Our contribution lies primarily in finding a combination that fits our application. Nonetheless, the derivation scheme discussed in III-C and the "lizard" encoding scheme from IV appear to be technical novelties. We believe that the security, privacy, verifiability, and especially the simplicity offered by our combination may certainly find application to other situations as well.

b) Demonstrator: An interactive demonstration of the most important features of PEP3 can be found at https://vwdata-p3.github.io/demo.html. Its source code is available at https://github.com/vwdata-p3/webdemo.

c) This paper is organised as follows: In Section II we describe the basic functioning of the transcryptor as a single entity, and then built it from five peers in Section III. We end the paper with the technical issue of encoding of IP addresses as points on curve25519 in IV.

II. Transcryptor

A. Polymorphic Encryption and Pseudonymisation

a) The group: We base PEP3 on curve25519, an elliptic curve introduced by Daniel Bernstein in [8], and named after the prime $p := 2^{255} - 19$, because it is both fast, and has stood up to the scrutiny caused by its popularity. Using Mike Hamburg's decaf-technique[9] curve25519 gives rise to the ristretto255[10] group $\mathbb{G}$, a special way to represent the cyclic group $\mathbb{Z}/\ell\mathbb{Z}$ of integers modulo the prime

$$\ell := 2^{252} + 27{,}742{,}317{,}777{,}372{,}353{,}535{,}851{,}937{,}790{,}883{,}648{,}493.$$

Although $\mathbb{Z}/\ell\mathbb{Z}$ and $\mathbb{G}$ are isomorphic as groups, the number 1 of $\mathbb{Z}/\ell\mathbb{Z}$ being send to a specific non-zero *base point* B, there is no known efficient algorithm to find given an element A from $\mathbb{G}$ the unique number n of $\mathbb{Z}/\ell\mathbb{Z}$ with $nB = A$. In other words, the *discrete log problem* is difficult in $\mathbb{G}$. We also assume it to be hard to solve the more difficult *decisional Diffie–Hellman problem* (see [11]) in $\mathbb{G}$, that is, to determine whether a triplet (A, M, N) of elements of $\mathbb{G}$ is a so-called *Diffie–Hellman triplet*, that is, whether writing $(A, M, N) \equiv (aB, mB, nB)$ we have $am = n$. We will see that such hardness assumptions can be used to show that cryptosystems based on $\mathbb{G}$ are resistant to several simple attacks. The actual security provided by these cryptosystems is, however, much more difficult to capture formally, see Section 3 "Security" of [8].

b) IP addresses: For now we will represent an IP address by a non-zero element A of the group $\mathbb{G}$. We will show in Section IV how to encode 128 bit IP addresses (thus supporting both IPv6 and IPv4) as elements of $\mathbb{G}$.

c) ElGamal Encryption: For the encryption of IP addresses we use the following scheme based on [4]. For any scalar $r \in \mathbb{Z}/\ell\mathbb{Z}$ the triple $(\, rB,\, M + rsB,\, sB\,)$ represents the encryption of a message $M \in \mathbb{G}$ for the *private key* $s \in \mathbb{Z}/\ell\mathbb{Z}$. The *public key* associated with s is the element sB of $\mathbb{G}$. In general, a *cyphertext* is a triple (β, γ, τ) of elements of $\mathbb{G}$, where β is the *blinding*, γ is the *core*, and τ is the *target*.[3] To encrypt a message $M \in \mathbb{G}$ for a public key $\tau \in \mathbb{G}$, one picks a random scalar $r \in \mathbb{Z}/\ell\mathbb{Z}$, and computes $\mathscr{E}(M, \tau, r) := (\, rB,\, M + r\tau,\, \tau\,)$. To decrypt a cyphertext (β, γ, τ) one computes $\mathscr{D}(\,(\beta, \gamma, \tau),\, s\,) := \gamma - s\beta$ given a private key $s \in \mathbb{Z}/\ell\mathbb{Z}$. Note that $\mathscr{D}(\mathscr{E}(M, \tau, r), s) = M + r(\tau - sB)$ equals M when $\tau = sB$. On the other hand, if $\tau \neq sB$, and r is chosen randomly, $\mathscr{D}(\mathscr{E}(M, \tau, r), s)$ could be any element of $\mathbb{G}$.

Note also that a triple (β, γ, τ) is the cyphertext of some message M if and only if $(\beta, \tau, \gamma - M)$ is a Diffie–Hellman triplet. Conversely, a triplet $(A, M, N) \in \mathbb{G}^3$ is a Diffie–Hellman triplet iff $(A, N, M) = \mathscr{E}(0, M, r)$ for some $r \in \mathbb{Z}/\ell\mathbb{Z}$. So deciding whether a triple (β, γ, τ) is the cyphertext for some message M (without additional information such as the secret key) is thus just as difficult as the decisional Diffie–Hellman problem. Decrypting a cyphertext without additional information might be even harder.

d) Pseudonyms: A *pseudonym* for an IP address represented by a non-zero group element $A \in \mathbb{G}$ is simply nA, where $n \in \mathbb{Z}/\ell\mathbb{Z}$ is a scalar called the *pseudonym key*. Note that depending on the pseudonym key, the pseudonym for A could be any element of $\mathbb{G}$.

In PEP3, the transcryptor keeps track of an *encryption key* s_P and a *pseudonym key* n_P, both non-zero scalars from $\mathbb{Z}/\ell\mathbb{Z}$, for every party P. The encryption key s_P is shared with P, while the transcryptor keeps n_P to herself. The *pseudonym for* P of an IP address $A \in \mathbb{G}$ is then $n_P A$, and an *encrypted pseudonym for* P of A is a triple of the form $(rB, n_P A + rs_P B, s_P B) \equiv \mathscr{E}(n_P A, s_P B, r)$ for some scalar $r \in \mathbb{Z}/\ell\mathbb{Z}$.

e) Translation: The transcryptor has the following three elementary operations on cyphertexts at her disposal, where $s, n, r \in \mathbb{Z}/\ell\mathbb{Z}$, cf. [3].

$$\begin{aligned} \textit{rekeying:} \quad & \mathscr{K}_s(\beta, \gamma, \tau) := (\, s^{-1}\beta,\, \gamma,\, s\tau\,) \\ \textit{reshuffling:} \quad & \mathscr{S}_n(\beta, \gamma, \tau) := (\, n\beta,\, n\gamma,\, \tau\,) \\ \textit{rerandomisation:} \quad & \mathscr{R}_r(\beta, \gamma, \tau) := (\, \beta + rB,\, \gamma + r\tau,\, \tau\,) \end{aligned}$$

To translate an encrypted pseudonym for party P to an encrypted pseudonym for party Q, the transcryptor applies $\mathscr{K}_{s_Q s_P^{-1}} \mathscr{S}_{n_Q n_P^{-1}} \mathscr{R}_{r'}$, where r' is some random scalar from $\mathbb{Z}/\ell\mathbb{Z}$. Note that $\mathscr{K}_{s_Q s_P^{-1}}$ changes the target of the encryption, sending $\mathscr{E}(M, s_P B, r)$ to $\mathscr{E}(M, s_Q B, s_P s_Q^{-1} r)$,

[3]We include the target τ in the cyphertext so that we can define the rerandomisation operation, $\mathscr{R}_r$, below.

while $\mathscr{S}_{n_Q n_P^{-1}}$ changes the target of pseudonyms, sending $\mathscr{E}(n_P M, \tau, r)$ to $\mathscr{E}(n_Q M, \tau, n_Q n_P^{-1} r)$.

The purpose of $\mathscr{R}_{r'}$ is more technical, and threefold. To begin, it prevents spoofing of the target in the cyphertext: if the triple $(rB, M + rsB, s'B)$, which pretends to be a cyphertext intended for $s'B$, but is actually decryptable by sB, is rerandomised, the result $(\,(r+r')B,\ M+rsB+r's'B,\ s'B\,)$ does not reveal M to someone not knowing s'. It also prevents a party P from obtaining the unencrypted pseudonym for Q of an IP address A by sending $(0, n_P A, s_P B)$ to the transcryptor for translation from P to Q. Finally, it makes the translation operation non-deterministic, reducing the risk of linkability.

f) (De)pseudonymisation: To translate a for party P encrypted IP address to a for party Q encrypted pseudonym, the transcryptor applies $\mathscr{K}_{s_Q s_P^{-1}} \mathscr{S}_{n_Q} \mathscr{R}_{r'}$, where r' is some random scalar. Depseudonymisation is performed similarly.

g) Polymorphism and homomorphism: The (ElGamal) encryption scheme we use is 'polymorphic' in the sense that a message encrypted for one party can be rekeyed to be decryptable by another party (without the need for intermediate decryption). Pseudonymisation is polymorphic in a similar sense. The fact that the translation between pseudonyms can be performed on cyphertext makes the encryption 'homomorphic' with respect to this translation operation. At this point the transcryptor, knowing the secret keys of the parties, can sidestep the polymorphism and homomorphism by first decrypting, then translating, and finally encrypting again. However, when the transcryptor is split into five peers in the next section, this trick is no longer possible, and the advantage of the polymorphic and homomorphic aspects of the encryption become clear.

B. Parties

Before we explain the way the transcryptor is built (from five peers), we will sketch how we intend her services be used by the different parties.

a) Metering and storage: The two most basic parties to PEP3 are the *metering process (MP)* generating flow records, and the *storage facility (SF)* storing the flow records. Recall that both parties get their own encryption keys s_{MP} and s_{SF} from the transcryptor, respectively. After the metering process has produced a batch of flow records aggregated from packets going over the network, it encrypts the associated IP addresses in these flow records using its own encryption key, s_{MP}, and sends them to the transcryptor for translation to encrypted pseudonyms for s_{SF}, which the transcryptor returns to the metering process. The metering process replaces the IP addresses in the flow records by these encrypted pseudonyms, and sends them along to the storage facility. Note that the metering process does not learn the pseudonyms for the storage facility, since they are returned by the transcryptor to the metering process encrypted for the storage facility's key, s_{SF}. The storage facility, having received and decrypted the encrypted pseudonyms in the flow records, stores the pseudonymised flow records in its database.

b) Retrieval: A party wishing to consult the records held by the storage facility may form a query in terms of their own set of pseudonyms, and then replace their pseudonyms by corresponding encrypted pseudonyms for the storage facility obtained from the transcryptor. Having received and performed the query, the storage facility returns the result, but only after having encrypted the pseudonyms (from the storage facility's set) with its encryption key, s_{SF}. Having received the flow record with encrypted IP addresses, the querying party consults the transcryptor again, this time to translate the encrypted pseudonyms from the storage facility's set to its own set.

c) Authorisation: To prevent free translation between pseudonyms and IP addresses (defeating the pseudonymisation) a specific permit (signed by some predetermined certification authority) could be required by the transcryptor for a party wishing to perform a translation or (de)pseudonymisation. For example, the metering process only could be given a personal permit to pseudonymise into storage facility pseudonyms. A party wishing to retrieve records from the storage facility, such as a researcher, would need a permit to translate pseudonyms from its own set to the set of the storage facility (and back). Note that if such a researcher was to collude with the metering process, they could link IP addresses with their own pseudonyms.

III. Five Peers

A. Ten Shares

In PEP3, the transcryptor is split into five peers, named A, B, C, D, and E. As a general rule, three out of five peers should be able to act as transcryptor. To this end, each pseudonym key n_P for a party P, is defined to be a product $n_P \equiv n_P^{\mathrm{ABE}}\, n_P^{\mathrm{ABC}}\, n_P^{\mathrm{BCD}}\, n_P^{\mathrm{CDE}}\, n_P^{\mathrm{ADE}}\, n_P^{\mathrm{ACD}}\, n_P^{\mathrm{BDE}}\, n_P^{\mathrm{ACE}}\, n_P^{\mathrm{ABD}}\, n_P^{\mathrm{BCE}}$ of $10 \equiv \binom{5}{3}$ *shares*, one for each triple of peers. Of course, the share n_P^{ABE} is given to the peers A, B, and E, and so on. Note that no two peers (such as D and E) have access to all shares (D and E do not have the share of ABC.) However, every triple does have access to all shares, because any two triples drawn from five peers must have at least one peer in common. The encryption key s_P for a party P is split similarly into ten shares. For brevity's sake, let $\mathcal{T} := \{\,$ABE, ABC, BCD, CDE, ADE, ACD, BDE, ACE, ABD, BCE$\,\}$ denote the set of all ten triples of peers.

a) Translation: An encrypted pseudonym for a party P can be translated to an encrypted pseudonym for party Q by applying the operations $\mathscr{K}_{s_Q^T (s_P^T)^{-1}} \mathscr{S}_{n_Q^T (n_P^T)^{-1}} \mathscr{R}_{r^T}$, where r^T is a random scalar, and T ranges over $\mathcal{T}$, in sequence. The order in which these operations are performed does affect the (random component of the) cyphertext, but not the resulting pseudonym (if the input was valid). Naturally, any of the three peers in the triple T can perform this operation.

A translation can also be performed by three operations instead of ten, as follows. Choose three peers, say A, C, and D, and split the triples among them, by, say,

$$\mathcal{T}_A := \{\,\text{ABE, ABC, ADE, ACD, ACE, ABD}\,\}$$
$$\mathcal{T}_C := \{\,\text{BCD, CDE, BCE}\,\} \quad \mathcal{T}_D := \{\,\text{BDE}\,\}.$$

Then define, for each $X \in \{A, C, D\}$,

$$n_P^X := \prod_{T \in \mathcal{T}_X} n_P^T \quad \text{and} \quad s_P^X := \prod_{T \in \mathcal{T}_X} s_P^T,$$

and have A, C, and D perform the operations $\mathscr{K}_{s_Q^X (s_P^X)^{-1}} \mathscr{S}_{n_Q^T (n_P^X)^{-1}} \mathscr{R}_{r^X}$ for all three $X \in \{A, C, D\}$, in sequence, on the encrypted pseudonym for P. (De)pseudonymisation can be performed similarly.

b) Alternative constellations: Our choice to divide the secrets of the transcryptor over the triples drawn from five peers is to some extend arbitrary. We could instead have chosen a system where the secrets are, for example, shared among pairs drawn from three peers (which is not resistant against collusion of two peers.)

B. Verification

Note that if one peer is offline, the transcryptor still functions. Nevertheless, a single peer can presently disrupt the system in another way, by producing erroneous results, possibly without being detected. One might argue that it is possible to prevent this by having multiple peers perform the same operation, and compare the results. This comparison is, however, complicated by the random component in the encryption. We propose a different method of verifying the peers' operations, namely by having the peers attach non-interactive[12] Schnorr type[13] proofs of correctness to their results. To keeps things simple we create these proofs from the following basic building block.

1) Certified Diffie–Hellman triplets: Recall that it is considered infeasible in general to determine whether a triplet (A, M, N) of group elements of $\mathbb{G}$ is a *Diffie–Hellman triplet*, that is, whether writing $(A, M, N) \equiv (aB, mB, nB)$ we have $am = n$. If the scalar a is known, however, the matter is easily settled by checking whether $aM = N$. We will describe a method by which a *prover* knowing a can prove to a *verifier* that (A, M, N) is a Diffie–Hellman triplet, without revealing a, using two group elements R_M, R_B, and one scalar s. We will say that (A, M, N) is *certified* by (R_M, R_B, s).

a) Creating the proof (R_M, R_B, s): The prover picks a random scalar $r \in \mathbb{Z}/\ell\mathbb{Z}$, and defines $R_B := rB$, $R_M := rM$, and $s := r + ha$, where $h := \mathrm{Hash}(A, M, N, R_M, R_B)$ for some appropriate hash function $\mathrm{Hash}\colon \mathbb{G}^5 \to \mathbb{Z}/\ell\mathbb{Z}$.

b) To verify a proof (R_M, R_B, s): for the alleged Diffie–Hellman triplet (A, M, N), the verifier computes $h := \mathrm{Hash}(A, M, N, R_M, R_B)$, and checks whether

$$sB = R_B + hA \quad \text{and} \quad sM = R_M + hN. \tag{1}$$

c) Infeasibility of fraud: To deceive the verifier into believing a triplet (A, M, N) is a Diffie–Hellman triplet verified by (R_M, R_B, s) a deceiver needs to solve the two equations in (1). Since the value of the hash h changes erratically depending on its inputs, it might as well be any scalar as far as the deceiver is concerned. Solving (1) can thus be considered a game, in which the deceiver first chooses A, M, N, R_M, R_B, then gets a random h as challenge, and must finally pick s such that the two equations in (1) hold. Having chosen A, M, N, R_M, R_B, the deceiver has a winning strategy if and only if she has a function $s\colon \mathbb{Z}/\ell\mathbb{Z} \to \mathbb{Z}/\ell\mathbb{Z}$ with, for all $h \in \mathbb{Z}/\ell\mathbb{Z}$,

$$s(h)B = R_B + hA \quad \text{and} \quad s(h)M = R_M + hN. \tag{2}$$

Taking $h = 0$, we see that $s(0)B = R_B$ and $s(0)M = R_M$. Substituting this result back into (2) for all $h \neq 0$,

$$\tfrac{s(h)-s(0)}{h} B = A \quad \text{and} \quad \tfrac{s(h)-s(0)}{h} M = N.$$

Hence (A, M, N) is indeed a Diffie–Hellman triplet.

d) Acknowledgement: The certified Diffie–Hellman triplets are essentially the same as the non-interactive version of protocol in Figure 5.7 of [14].

2) Certifying $\mathscr{K}_s \mathscr{S}_n \mathscr{R}_r$: Such certified Diffie–Hellman triplets can be used by a peer wishing to show to a party (via a non-interactive Schnorr type proof) that a triple (β', γ', τ') is the result of performing the operation $\mathscr{K}_s \mathscr{S}_n \mathscr{R}_r$ on a triple (β, γ, τ), that is, that $(\beta', \gamma', \tau') = (ns^{-1}(\beta + rB),\ n(\gamma + r\tau),\ s\tau) \equiv \mathscr{K}_s \mathscr{S}_n \mathscr{R}_r(\beta, \gamma, \tau)$. Indeed, the peer would need only certify the five Diffie–Hellman triplets

$$(\ ns^{-1}B,\ \beta + rB,\ \beta'\ ), \quad (\ nB,\ \gamma + r\tau,\ \gamma'\ ), \quad (sB, \tau, \tau'),$$
$$(\ sB,\ ns^{-1}B,\ nB\ ), \quad \text{and} \quad (rB, \tau, r\tau).$$

It's assumed here that the party can know what sB and nB should be. If, as in the case of translation of a pseudonym, s is a composite, such as $s \equiv s_P^T (s_Q^T)^{-1}$, then additional proof must be provided for this to the peer, for instance a certified Diffie–Hellman triplet $(\ s_P^T (s_Q^T)^{-1} B,\ s_Q^T B,\ s_P^T B\ )$ in our example.

C. Derivation of pseudonym and encryption keys

A triple T of peers derives the pseudonym key n_P^T and encryption key s_P^T for a party P from a *master pseudonym key* n^T, and a *master encryption key* s^T, respectively. We thereby circumvent the troubles of having to generate, store and synchronise keys s_P^T and n_P^T for every new party P, on demand. The keys are derived as follows: assuming that each party P has some unique identifier id_P from some set $\mathcal{I}$, and given a hash function $H\colon \mathcal{I} \to \mathbb{Z}/(\ell - 1)\mathbb{Z}$, we set

$$n_P^T := (n^T)^{H(\mathrm{id}_P)} \quad \text{and} \quad s_P^T := (s^T)^{H(\mathrm{id}_P)}.$$

We derive the keys in this particular manner in order to make it possible for a peer to give proof that $n_P^T B$ was derived from $n^T B$, using the 253 group elements $n^T B$, $(n^T)^2 B$, $(n^T)^4 B$, $\ldots$, $(n^T)^{2^{252}} B$, by, writing $H(\mathrm{id}_P) = \sum_{k=1}^n 2^{i_k}$ where $0 \leq i_1 < i_2 < \cdots < i_n \leq 252$, certifying the Diffie–Hellman triplets

$$\begin{array}{c}
(\ (n^T)^{2^{i_1}} B,\ (n^T)^{2^{i_2}} B,\ (n^T)^{2^{i_1} + 2^{i_2}} B\ ) \\
(\ (n^T)^{2^{i_1} + 2^{i_2}} B,\ (n^T)^{2^{i_3}} B,\ (n^T)^{2^{i_1} + 2^{i_2} + 2^{i_3}} B\ ) \\
\vdots \\
(\ (n^T)^{2^{i_1} + \cdots + 2^{i_{n-1}}} B,\ (n^T)^{2^{i_n}} B,\ n_P^T B\ ).
\end{array}$$

Such a proof for $n_P^T B$ is needed by peers and parties wishing to check a proof (from Section III-B) in which $n_P^T B$ appears.

Any party Q should be able to request such a proof for $n_P^T B$ from a peer in the triple T. In particular, the party P can pass along a proof of $n_P^T B$ to a peer not in T needing proof of $n_P^T B$ to verify, for example, a depseudonymisation request. In this way, the peer does not need to contact the other peers.

Regarding the security of this derivation scheme: we conjecture that recovering n^T from the group elements $(n^T)^{2^0}B, \ldots, (n^T)^{2^{252}}B$ is essentially as difficult as computing the discrete log for one of 253 random group elements.

D. Setup and enrolment

We assume that the peers and parties can authenticate one another and communicate securely, *e.g.,* by using TLS and certificates. To start PEP3 each triple T of peers needs to decide on secrets n^T and s^T, and the public parts $n^T B$, $(n^T)^2 B, \ldots$ and $s^T B$, $(s^T)^2 B, \ldots$ need to be shared with the other peers. How this setup could be performed reliably and verifiably is beyond the scope of this short paper. To add a party P to the system, by a process called *enrolment*, P simply requests the public keys $n^T B$, $(n^T)^2 B, \ldots$ and $s^T B$, $(s^T)^2 B$, $\ldots$ from every peer, and the secret s_P^T from every peer in the triple T, with proof for $s_P^T B$ from $s^T B$, $(s^T)^2 B, \ldots$. If two peers in a triple send incorrect values for s_P^T, then if the other three peers are honest, P can detect the correct value for s_P^T, and thus which peers were dishonest, by following the majority's claim for the value of $s^T B$, $(s^T)^2 B, \ldots$.

IV. Encoding IP addresses

One technical problem we encountered when using the ristretto255 group $\mathbb{G}$ was the lack of a direct way to encode a 128 bit piece of data w (such as an IPv6 or IPv4 address) as an element $\underline{w}$ of $\mathbb{G}$ in such a way that the data w can cheaply be recovered from $\underline{w}$. Such an encoding is useful for encrypting w using the ElGamal scheme described in Section II-A.

The other direction presents no problem: there is a canonical and reversible way to encode an element of $\mathbb{G}$ as a 32-byte string, but only $\ell/2^{256} \approx 6.25\%$ of all 32-bytes strings are a valid encoding of an element from $\mathbb{G}$. So what is usually done (circumventing the need to encode a message as group element before encrypting it) is to pick a random group element and use its 32-byte encoding to encrypt the message *symmetrically*.

This solution is not viable for our system, because rerandomisation and reshuffling cannot be applied to the symmetric cyphertext. Instead we would like to use *elligator 2*[15], which does give a reversible map $\mathsf{ell2}\colon \mathbb{Z}/p\mathbb{Z} \longrightarrow \mathbb{G}$, but each element of $\mathbb{G}$ can have up to 16 preimages under ell2. Since $\mathsf{ell2}(x) = \mathsf{ell2}(-x)$ we can discard half the preimages by considering only the elements of $\mathbb{Z}/p\mathbb{Z}$ whose minimal positive representative is even. Thus $\mathsf{ell2}'\colon \{0,1\}^{253} \to \mathbb{G}$ defined by $\mathsf{ell2}'(b_1 \cdots b_{253}) := \mathsf{ell2}(\, b_1 2^1 + b_2 2^2 + \cdots + b_{253} 2^{253}\,)$ is reversible, and the preimage $\mathsf{ell2}'^{-1}(A)$ of an element $A \in \mathbb{G}$ has at most 8 elements. Now, define $\mathsf{lizard}\colon \{0,1\}^{128} \to \mathbb{G}$ by[4] $\mathsf{lizard}(b_1 \cdots b_{128}) := \mathsf{ell2}'(b_1 \cdots b_{128} h_1 \cdots h_{125})$, where $h_1 \cdots h_{125}$ are the first 125 bits of the SHA-256 hash of $b_1 \cdots b_{128}$. Then lizard is easily computable, and reversible, and, the preimage $\mathsf{lizard}(A)$ of an element $A \in \mathbb{G}$ almost always contains at most 1 element. Indeed, assuming that the bits of a word w in the preimage $\mathsf{ell2}'(A)$ are distributed randomly, the chance that the last 125 bits of such a word match the first 125 bits of the SHA-256 of w should be $\frac{1}{2^{125}}$. Thus the chance that given $w \in \{0,1\}^{128}$ the preimage $\mathsf{lizard}^{-1}(\mathsf{lizard}(w))$ contains only w is at least $(1 - \frac{1}{2^{125}})^7$. So even if 10^{10} computers would apply lizard to 10^{10} unique IP addresses$/s$ for 300 years ($\approx 10^{10}\, s$), all $\approx 2^{100}$ IP addresses will map to a group element with a unique preimage with probability of at least $(1 - \frac{1}{2^{125}})^{7 \cdot 2^{100}} \geq 1 - 7 \cdot 2^{100} \cdot \frac{1}{2^{125}} \geq 1 - \frac{1}{2^{21}} \geq \frac{999{,}999}{1{,}000{,}000}$.

[4]Implementations of (a trivial variation on) lizard can be found on https://github.com/vwdata-p3/webdemo/blob/master/ed25519.py and https://github.com/bwesterb/go-ristretto/blob/master/ristretto.go.

V. Conclusion

We have described PEP3, a system for pseudonymising IP flow data built on curve25519 via the ristretto255 group $\mathbb{G}$. An important feature of PEP3 is a robust transcryptor (consisting of five peers) that functions even when two peers act dishonestly. Moreover, the peers do not learn the pseudonyms they process, and the the peers' actions can be verified.

References

[1] B. Claise, "Cisco systems netflow services export version 9," RFC 3954, October 2004, doi: 10.17487/rfc3954.

[2] B. Claise, B. Trammell, and P. Aitken, "Specification of the IP flow information export (IPFIX) protocol for the exchange of flow information," RFC 7011, September 2013, doi: 10.17487/rfc7011.

[3] E. Verheul, B. Jacobs, C. Meijer, M. Hildebrandt, and J. de Ruiter, "Polymorphic encryption and pseudonymisation for personalised healthcare," Cryptology ePrint Archive Report 411, 2016, https://eprint.iacr.org/2016/411.

[4] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985, doi: 10.1109/TIT.1985.1057074.

[5] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," in *EUROCRYPT'98*, ser. LNCS, vol. 1403, 1998, pp. 127–144, doi: 10.1007/BFb0054122.

[6] J. Camenisch and A. Lehmann, "(un)linkable pseudonyms for governmental databases," in *CCS'15*, 2015, pp. 1467–1479, doi: 10.1145/2810103.2813658.

[7] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979, doi: 10.1145/359168.359176.

[8] D. J. Bernstein, "Curve25519: new Diffie-Hellman speed records," in *PKC 2006*, ser. LNCS, vol. 3958, 2006, pp. 207–228, doi: 10.1007/11745853_14.

[9] M. Hamburg, "Decaf: Eliminating cofactors through point compression," in *CRYPTO 2015*, ser. LNCS, vol. 9215, 2015, pp. 705–723, doi: 10.1007/978-3-662-47989-6_34.

[10] H. de Valence, "The ristretto group [homepage]," https://ristretto.group, accessed: April 28 2019.

[11] D. Boneh, "The decision Diffie–Hellman problem," in *ANTS 1998*, ser. LNCS, vol. 1423, 1998, pp. 48–63, doi: 10.1007/BFb0054851.

[12] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *CRYPTO'86*, ser. LNCS, vol. 263, 1986, pp. 186–194, doi: 10.1007/3-540-47721-7_12.

[13] C.-P. Schnorr, "Efficient identification and signatures for smart cards," in *CRYPTO'89*, ser. LNCS, vol. 435, 1989, pp. 239–252, doi: 10.1007/0-387-34805-0_22.

[14] B. Schoenmakers, "Lecture notes [on] cryptographic protocols," February 2019, version 1.4, available at https://www.win.tue.nl/~berry/CryptographicProtocols/LectureNotes.pdf.

[15] D. J. Bernstein, M. Hamburg, A. Krasnova, and T. Lange, "Elligator: Elliptic-curve points indistinguishable from uniform random strings," in *CCS'13*, 2013, pp. 967–980, doi: 10.1145/2508859.2516734.