Privacy Preserving Measurement
Internet-Draft
Intended status: Informational
Expires: 21 August 2025

Y. Wang
W. Chang
Y. Lu
C. Hong
J. Peng
Ant Group

17 February 2025

PSI based on ECDH
draft-wang-ppm-ecdh-psi-01

Abstract

This document describes Elliptic Curve Diffie-Hellman Private Set Intersection (ECDH-PSI) for 2 participants. It instantiates the classical Meadows match-making protocol with standard elliptic curves and hash-to-curve methods, enabling the participants to find common records in a privacy-respecting way. In ECDH-PSI, records are encoded to points on an elliptic curve, and masked by the private keys of both participants. To compute the intersection, a participant only uses the jointly masked datasets and its original dataset locally, which prevents its partner from learning the private records not in the intersection.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at https://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on 21 August 2025.

Copyright Notice

Table of Contents

1. Introduction

Private Set Intersection (PSI) schemes enable the discovery of shared elements among different parties' datasets without revealing individual data. They are widely used to identify overlapping data elements between two or more parties while preserving the confidentiality of each party's original data. PSI is one of the most frequently used privacy preserving techniques in business. It enables a user to detect whether his/her password is leaked without giving away the password to the server [MS21], and allows multiple companies to find their common customers without revealing raw data. Furthermore, many privacy-respecting systems can leverage PSI schemes to collaborate with other data providers for the purpose of enhancing the privacy of data-exchange processes. As an example, a service provider who has aggregated attributes from individuals following [I-D.ietf-ppm-dap][draft-irtf-cfrg-vdaf-12] can use PSIs to compare its results with another entity without disclosing any attribute not owned by the partner.

The classical Diffie-Hellman PSI [Meadows86] has been published for more than thirty years. The academic has also proposed numerous PSI schemes with new functionalities and secure guarantees, such as [KKRT16][CHLR18][RR22], but DH-PSI is still preferable due to its simplicity and communication efficiency. This document describes a widely deployed instance of DH-PSI denoted by Elliptic Curve Diffie-Hellman PSI (ECDH-PSI). Compared with the finite field parameters used in the classical version, elliptic curves are commonly considered to be more efficient and secure [IMC][LOGJAM], and are extensively adopted by recent standards including [RFC8031][RFC8446][RFC9497].

This document describes 2-party ECDH-PSI as a two-phase protocol, including a handshake phase which negotiates parameters such as cipher suites and a data exchange phase which transfers masked datasets. At the end of data exchange phase, one or both parties output the intersection result.

1.1. Requirements Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14 [RFC2119] [RFC8174] when, and only when, they appear in all capitals, as shown here.

1.2. Data Structures and Notations

In this document, data structures are defined and encoded according to the conventions laid out in Section 3 of [RFC8446].

The following notations are used throughout this document:

*sqrt(x): The square root of x.

*log(x,y): The logarithm of y to the base x.

*x^-1: The inverse of x (over a finite field).

*x||y: Concatenate string x and y.

*intersection(set_1,set_2): The intersection of set_1 and set_2

*|set|: The cardinality of set.

2. Background

This section gives brief introductions for elliptic curves, hash-to-curve methods and the Transport Layer Security (TLS) protocol.

2.1. Elliptic Curves

An elliptic curve E is defined by a two-variable equation over a finite field F with prime characteristic p larger than three. Except for a special point called point at infinity, a point on E is a (x,y)-coordinate pair that satisfies the curve equation, where x and y are elements of F. All distinct points of curve E constitute an algebraic group of order n, where the point at infinity is the identity element. Applications and standards generally use a prime order (sub)group <G> generated by a point G on E. The order of <G> is denoted by r.

This document uses term "addition" to refer to the group operation of an elliptic curve group, meaning two points P and Q can be added together to get a third point R. Scalar multiplication refers to the operation of adding a point P with itself repeatedly. This document uses scalar_mul to denote the scalar multiplication process. It takes an integer x<r and a point P as inputs and outputs the multiplication result Q, which is written by Q=scalar_mul(P,x) or simply Q=P^x. When set_p is a set of EC points, set_m=set_p^x denotes that all elements of set_p are multiplied by x to obtain set_m.

Many cryptographic applications require the participants to generate elliptic curve key pairs. Usually, a key pair refers to a (PK,sk) tuple, where sk is an integer generated uniformly between 1 and r-1, and PK=scalar_mul(G,sk). In the context of ECDH-PSI, only sk, namely the private key, is used. Thus, this document uses the notation sk=keygen() to refer to the process of generating a private key on the elliptic curve and omits PK.

2.2. Hashing to Elliptic Curves

A hash_to_curve function [RFC9380] encodes data of arbitrary length to a point on an elliptic curve. The encoding process first hashes the byte string to elements of fixed length, and then calculates a point on the curve with the elements using so-called map_to_curve and clear_cofactor functions.

[RFC9380] describes a series of hash_to_curve methods (which are also called "suites") for standard curves such as NIST P-256 and curve25519. It specifies two encoding types denoted by "encode_to_curve" and "hash_to_curve". Both encodings can be proven indifferentiable from random oracles (ROs)[MRH04], but have different output distributions. This document only uses "hash_to_curve" encodings which output uniformly random points. It also uses the notation set_p=hash_to_curve(set_d) to refer to the process of encoding every elements in set_d to points on a curve and obtains set_p.

[RFC9380] requires all uses of the hash_to_curve suites must enforce domain separation with domain separation tags (DSTs). In particular, DSTs are distinct strings that enable different applications to simulate distinct RO instances with the same hash function. To meet the requirement of DSTs, this document allocates each suite with a unique DST, and uses the notation of set_p=hash_to_curve(set_d) assuming that the corresponding DST strings have been taken as internal inputs.

A suite defined by [RFC9380] includes a series of parameters such as the curve, hash function, and encoding type. In applications, these suites are very efficient to be referred to, as the included parameters can also be used beyond the scenario of mapping data to curves. For example, the participants can negotiate a hash_to_curve suite, and then use the curve deduced from the suite for other cryptographic functions such as encryption and key establishment.

To extend the usage of hash_to_curve suites, [RFC9380] also describes methods and naming conventions for defining new suites, which enables developers to define suites with new curves and hash functions.

2.3. Transport Layer Security

The TLS protocol [RFC8446] has been widely used for secure communication over the Internet. It operates over any underlying communication channel that provides reliable, in-order data stream, ensuring important security properties of authentication, confidentiality and integrity. In particular, TLS provides mutual authentication methods so that the peers can authenticate each other. It also uses DH key exchange to establish session keys and employs authenticated encryption to guarantee that the transferred messages are encrypted and not manipulated.

The notion of "channel binding" is originally proposed by [RFC5056], which enables applications to bind authentication to secure sessions at lower layers in the network stack so as to prevent the so-called Man-In-The-Middle (MITM) attack. To extend the notion to TLS, [RFC9266] presents a method of binding applications to the underlying TLS 1.3 sessions, which is denoted by 'tls-exporter'. It outputs exported keying material (EKM) established by the TLS session, which can be seen as the output of a pseudorandom function (PRF) based on TLS. The application can bind to the TLS session by taking EKM as an input to the application-layer cryptographic session.

3. Protocol

We begin with a very brief description on how ECDH-PSI works. Firstly, the participants, A and B, agree on a group <G> along with the hash_to_curve suites, and generate fresh private keys over <G>. Then, both A and B convert their records to points over <G> and multiply the points with both parties' private keys for masking. Finally, they exchange the sets of masked elements, compute their intersection, and match the intersections with original datasets locally.

To prepare the ECDH-PSI protocol, A and B need to:

* Establish a TLS channel with mutual authentication.

* Negotiate the parameters used for subsequent steps, especially the hash_to_curve suite.

* Generate EC private keys with sk_A=keygen() and sk_B=keygen(). These keys are also denoted by "ECDH-PSI keys", and only used in current ECDH-PSI session.

After the preparation, assume that A has set_A, B has set_B, a simplified protocol flow of ECDH-PSI is shown by Figure 1, and explained step-by-step as follows:

1. Each participant maps its records to EC points with hash_to_curve, and masks the points locally with its own private key by scalar_mul. In this step, the ECDH-PSI session is bound to the underlying TLS session with the channel binding mechanism introduced by Section 2.3. We defer the details of session binding to Section 3.3.

2. A and B exchange their locally masked sets pset_A and pset_B.

3. Upon receiving the masked data from its partner, a participant masks the received points with its private key.

4. Each participant sends the jointly masked set (i.e., pset_BA and pset_AB) back to its partner.

5. Each participant calculates intersection of the set calculated in Step 3 and the set received in Step 4, matches the original data set with the intersection, and finally outputs the matching result.

In order to clarify the statement, this document uses "the first round" or "round 1" to refer to Step 2, and "the second round" or "round 2" for Step 4.

The classical description of ECDH-PSI enables both parties to output PSI results and thus requires two complete rounds (i.e. Step 2 and Step 4) to exchange masked data. However, many application scenarios require that only one participant (commonly, the participant who initiates the session) should output the intersection and the other gets nothing. In such scenarios, Step 4 and Step 5 are executed unidirectionally, where only one party sends jointly masked data in Step 4 and only the receiver of that message can output the result.

To output PSI result, the participants should also match the original data items with their corresponding masked EC points. In particular, the relationship can be established with unique indexes, where an original data record and corresponding masked EC point(s) are linked to the same index.

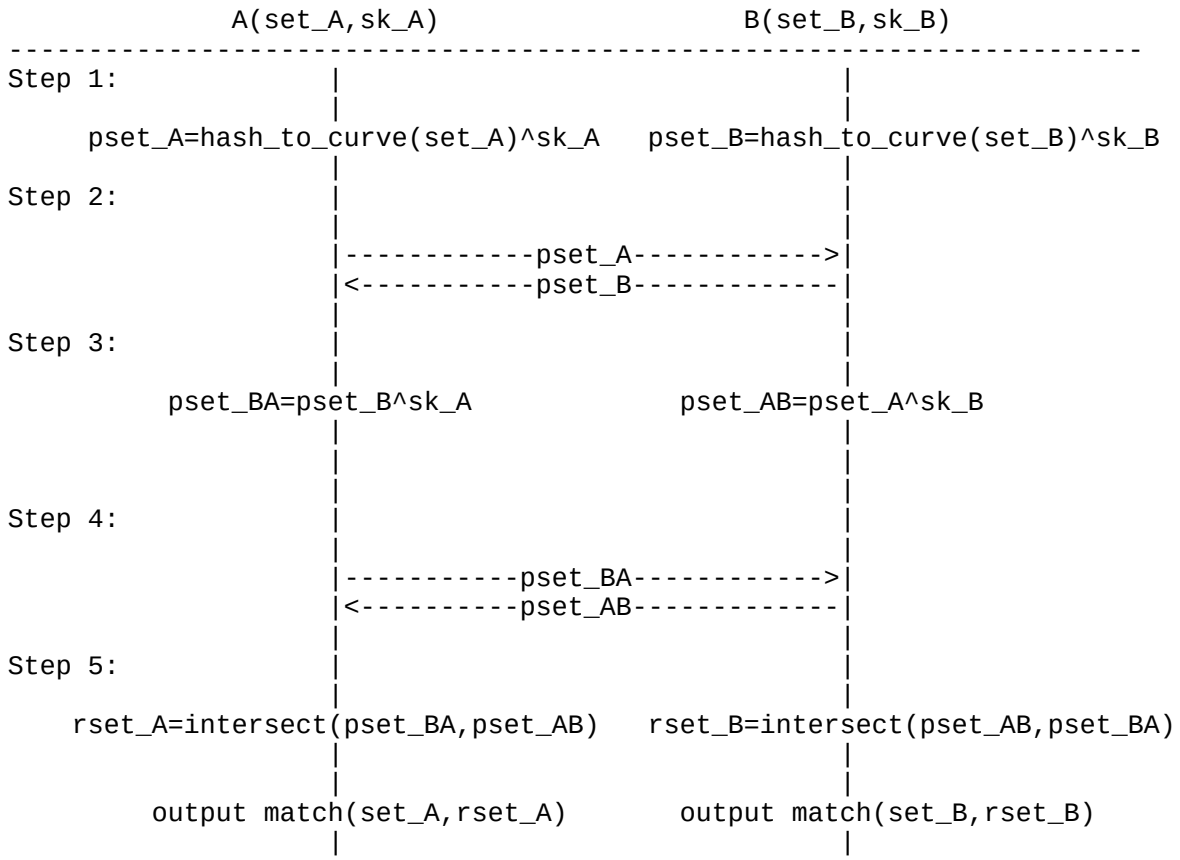

Figure 1: A Simplified Protocol Flow of ECDH-PSI

3.1. Overview

The specification of ECDH-PSI consists of two phases as shown in Figure 2, including a handshake phase which negotiates the protocol parameters and a data exchange phase that performs the operations described by Figure 1.

* In the handshake phase, a participant, which is denoted by requester, sends a HandshakeRequest message to initiate the ECDH-PSI protocol flow. This message carries lists of parameters supported by the requester. Then, the other participant, denoted by responder, selects parameters from the requester’s lists, and sends the parameters with its data information in a HandshakeResponse message.

* Next, in the data exchange phase, both parties perform ECDH-PSI operations with parameters selected in the handshake phase, where EcdhPsiBatch messages carrying masked EC points are exchanged.

In the second round, the EcdhPsiBatch message sent from the requester is optional since the requester may not allow the responder to output the intersection, which is determined in the handshake phase.

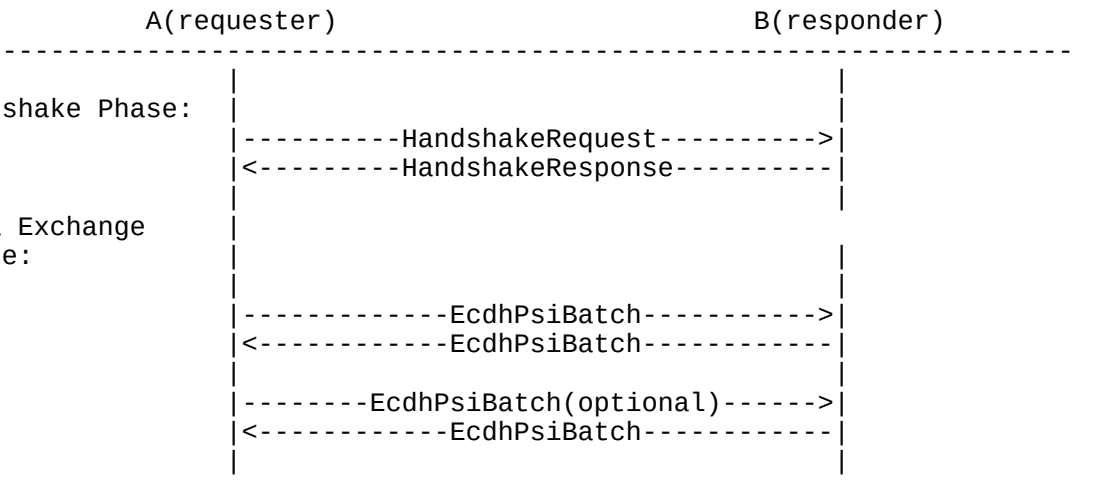

Figure 2: An Overview of the Two-Phase Specification of ECDH-PSI

3.2. Handshake

The handshake phase includes a HandshakeRequest and a HandshakeResponse message. The structures of these messages are documented as follows.

3.2.1. HandshakeRequest

Structure of HandshakeRequest:

```
struct {
  uint8 version;
  uint8 output_mode;
  uint64 record_num;
  ProtocolProposal protocol_param;
} HandshakeRequest;
```

version: The version of ECDH-PSI protocol. Currently only value 1 is allowed.

record_num: The number of requester’s data records.

output_mode: Whether the responder is allowed to output the result, which also decides whether the optional EcdhPsiBatch is sent in the second round. The meaning of OutputMode is explained as follows:

* 0: Both parties output the intersection result, which means EcdhPsiBatch are sent mutually in the second round.

* 1: Only the requester outputs PSI result, which means only the responder sends EcdhPsiBatch in the second round.

protocol_param: ECDH-PSI protocol configurations supported by the requester.

3.2.1.1. Protocol Proposal

The ProtocolProposal message describes ECDH-PSI protocol parameters supported by the requester.

Structure of this message:

```
enum {
  null (0),
  P256_XMD_SHA256_SSWU_NU_ (1),
  P384_XMD_SHA384_SSWU_NU_ (2),
  P521_XMD_SHA512_SSWU_NU_ (3),
  curve25519_XMD_SHA512_ELL2_NU_ (4),
  (255)
} HashToCurveSuite;

enum { compressed (0), uncompressed  (1), (255) } PointOctetFormat;

enum {
  no_truncation (0),
  128_bit_truncation (1),
  192_bit_truncation (2),
  (255)
} TruncationOption;

struct {
  HashToCurveSuite    suites<1..255>;
  PointOctetFormat    point_octet_formats<1..255>;
  TruncationOption    truncation_options<1..255>;
} ProtocolProposal;
```

suites: The list of supported HashToCurveSuite in order of requester’s preference.

The suites defined here are slightly different from [RFC9380]. To be compatible with the representation language of [RFC8446] where enumerate types are defined by C identifiers, the colons (":") are replaced with underscores ("_"), and hyphens ("-") in hash functions are removed (e.g., "SHA-256" is replaced by "SHA256").

Different suites use different DST strings as required by [RFC9380]. In particular, ECDH-PSI uses "ECDH-PSI-V<xx>-<suites>" as its DST, where <xx> is a two-digit hexadecimal number indicating the current protocol version as specified by the version field of HandshakeRequest, and <suites> is the ASCII string representation of the currently adopted suite as defined by [RFC9380]. As an example, when both participants agree to use the suite for P256 curve, the corresponding DST is "ECDH-PSI-V01-P256_XMD_SHA256_SSWU_NU_".

point_octet_formats: The list of octet string formats representing EC points, in the order of requester's preference. This field enables the participants to weigh the tradeoffs between the costs of communication and computation as discussed by Section 4.1.

* For the curves defined in [FIPS186-4], the PointOctetFormat values refer to the corresponding compressed/uncompressed formats documented by [ECDSA].

* Points on Curve25519 are always encoded following the description by Section 5 of [RFC7748] as 32 byte strings. Unlike the other curves, the scalar multiplication of curve25519 only relies on one 32-byte coordinate without the cost of decompressing to the (x,y)-coordinate representation, which eliminates the need of making a tradeoff.

truncation_options: Whether the requester supports truncation for data transferred in the second round. The options are listed in the requester's order of preference, where no_truncation MUST be included. The truncation could decrease the costs of data transmission and storage, and even accelerate the computation process of intersection, at the expense of a (negligible) false-positive rate.

In this document, only truncation options of 128 and 192-bit are allowed, with a restriction that the total number of data items owned by both participants SHOULD be no more than 2^40, ensuring that the probability of collision is smaller than 2^-48 when 128 bits are truncated. See Section 5.7 for a detailed discussion. The requester MUST set this field to a single no_truncation option when its data set size has already been equal or larger than 2^40.

The truncation process utilizes Key Derivation Functions (KDFs) as many key-exchange protocols that use shared secrets to derive shorter strings which can be seen as uniformly distributed cryptographic keys (e.g., [RFC8446][RFC9382]). In this document, the KDFs are instantiated by Hashed Message Authentication Code (HMAC) [RFC2104], along with HMAC-based KDF (HKDF)[RFC5869], and the hash function included in hash_to_curve suite.

Following [RFC5869], a KDF function kdf() takes a input keying material ikm, a salt value salt, an application specific information info and output length len as its inputs. Let mask_data be the string representation of a jointly masked point, a participant SHOULD truncate mask_data with:

truncated_mask_data = kdf(mask_data, nil, "ECDH-PSI", truncation_len)

In particular, the KDF takes mask_data as the input keying material and truncation_len of 128 or 192 depending on the selected truncation option. It does not take a salt value, uses ASCII string "ECDH-PSI" as the application specification information, and outputs the derived string as truncation result. When "ECDH-PSI" string is taken as the input for kdf, the terminating null byte SHOULD not be included.

3.2.2. HandshakeResponse

The responder sends HandshakeResponse to reply to a HandshakeRequest message. It includes selected parameters and the size of responder’s dataset.

Structure of this message:

```
enum {
  success(0),
  generic_error         (0x01),
  unsupported_version   (0x02)
  invalid_request       (0x03),
  out_of_resource       (0x04),
  unsupported_parameter (0x05),
  (0xFF)
} HandshakeStatus;

struct {
  HandshakeStatus status;
  uint64 record_num;
  ProtocolResult protocol_param;
} HandshakeResponse;
```

status: Indicating the status of handshake. The meanings of error statuses are listed as follows:

* generic_error: The error cannot be categorized to the rest error statuses defined by HandshakeStatus.

* unsupported_version: The responder does not support the ECDH-PSI protocol version given by version.

* invalid_request: The responder cannot parse the request correctly. The message may be in wrong format and cannot be parsed as a normal HandshakeRequest message.

* out_of_resource: The responder does not have enough resource to handle the ECDH-PSI process with the parameters and options provided by the requester.

* unsupported_parameter: The responder rejects all options proposed by one of the suggested option lists included in HandshakeRequest.

The responder MUST ignore all options or parameters that it cannot recognize when parsing the HandshakeRequest message. If one of the suggested option lists is filled with unrecognized parameters, it SHOULD reply with a HandshakeResponse carrying unsupported_parameter.

Upon receiving a HandshakeResponse message without success status, the requester MUST ignore the other fields included in this message and terminate the session.

item_num: The size of responder’s dataset.

protocol_param: The protocol parameter selected by the responder, including the hash_to_curve suite, EC point string format and truncation option.

3.2.2.1. Protocol Result

The structure of ProtocolResult is defined as follows:

```
struct {
  HashToCurveSuite    suite;
  PointOctetFormat    point_octet_format;
  TruncationOption    truncation_option;
} ProtocolResult;
```

suite: The hash_to_curve suite selected by the responder.

point_octet_format: The format of EC point octet string chosen by the responder.

truncation_option: The truncation option selected by the responder. The responder SHOULD consider the sum of both participants' dataset sizes. If the sum is larger than 2^40, truncation MUST not be used, which means the responder MUST set this field to no_truncation.

When deciding the fields of ProtocolResult, the responder SHOULD consider the requester's preference. That is to say, if multiple values of a list are acceptable for the responder, it SHOULD choose the topmost one.

3.2.3. Support More hash_to_curve Suites

Participants can negotiate hash_to_curve suites besides the ones listed by Section 3.2.1.1. They can use other suites defined by [RFC9380], or use a self-defined suite following the syntax and convention given by Section 8.9 and 8.10 of [RFC9380].

As an example, this document next defines a hash_to_curve suite for the ShangMi(SM) curve and cryptography algorithms, which have been accepted by international standards such as [ISO-SM2], [ISO-SM3] and [RFC8998]. However, if the participants decide to use elliptic curve parameter with security level less than 128 bit, or a curve over an extension field, they MUST evaluate the security of such parameter carefully as discussed by Section 5.

The new suite, denoted by curveSM2_XMD_SM3_SSWU_RO, encodes data to points on the so-called curveSM2[GBT.32918.5-2017] with SM3 hash algorithm[GBT.32905-2016], and reuses the expand_message_xmd and Simplified Shallue-van de Woestijne-Ulas (SSWU) methods for message expansion and mapping.

In particular, curveSM2_XMD_SM3_SSWU_RO_ is defined as follows:

* encoding type: hash_to_curve
* E: y^2 = x^3 + A * x + B, where
  - A = -3
  - B = 0x28e9fa9e9d9f5e344d5a9e4bcf6509a7f39789f515ab8f92ddbcbd414d940e93
* p: 2^256-2^224-2^96+2^64-1

* r: 0xfffffffeffffffffffffffffffffffff7203df6b21c6052b53bbf40939d54123

* m: 1

* k: 128

* expand_message: expand_message_xmd (Section 5.3.1 of [RFC9380])

* H: SM3

* L: 48

* f: SSWU method (Section 6.6.2 of of [RFC9380])

* Z: -9

* h_eff: 1

The value Z is calculated by the sage script given by Appendix H.2 of [RFC9380] with the p, A and B parameters of curveSM2.

The participants SHOULD agree on the meaning of HashToCurveSuite values. In this document, suite curveSM2_XMD_SM3_SSWU_RO_ is identified as follows:

```
enum {
  null                                (0),
  ... // (1) to (4) follow section 3.2.1.1
  curveSM2_XMD_SM3_SSWU_RO_           (5),
  ... // other hash_to_curve suites
  (255)
} HashToCurveSuite;
```

3.3. Data Exchange

In the data exchange phase, masked EC points are exchanged as EcdhPsiBatch messages. The participants MUST generate fresh ECDH-PSI keys after the handshake phase, and use the newly generated keys to mask the EC points. The ECDH-PSI keys SHOULD be removed immediately after the data exchange phase in order to prevent leakage or being reused by other applications by coincidence.

This phase consists of two rounds. In the first round, both participants send EcdhPsiBatch messages carrying locally masked EC points, where the requester sends the first one. In the second round, EcdhPsiBatch messages carrying jointly masked EC points are sent. If the requester sets output_mode field by 1, then only the responder sends an EcdhPsiBatch message, otherwise both participant sends EcdhPsiBatch messages as in the first round.

If the participants have negotiated a truncation_option of 128 or 192 bit in the handshake phase, such option MUST NOT be used in the first round, since truncated strings cannot be recovered to EC points, which makes the second mask operation completely infeasible.

3.3.1. EcdhPsiBatch

The structure of EcdhPsiBatch is defined as follows:

```
struct {
  uint64 index;
  opaque encoded_point[encoded_point_length];
} ECPointOctet;

struct {
  uint32        batch_type;
  uint64        batch_count;
  ECPointOctet  encoded_points<1..2^64-1>;
} EcdhPsiBatch;
```

batch_type: This field indicates the status of current batch.

* 0: Error occurs, which means that the current ECDH-PSI session MUST be terminated immediately.

* 1: The data included in the current batch are only masked by the owner’s private key.

* 2: The data has been jointly masked with both participants’ private keys.

The participant SHOULD check this field first before parsing encoded_points field. If batch_type is 0, or the value is not the same as expected, the receiver MUST discard all intermediate results, remove the ECDH-PSI key, and terminate the session.

batch_count: Number of EC points included in the current EcdhPsiBatch message. The receiver SHOULD also check that the value of this field is the same as expected and matches the data field. It MUST terminate the session if it is not the case, since an attacker may

send more points than expected so as to obtain some privilege of breaking the major privacy goal of ECDH-PSI. See Section 5.2 for more detail.

encoded_points: This field carries multiple EC points with ECPointOctet, which number is specified by the batch_count field. Each ECPointOctet structure includes an unique index allocated by the owner of the corresponding original data, and the octet string encoded_point encoding an EC point. The length of encoded_point, which is encoded_point_length, is determined by the hash_to_curve suite, the compression option and the truncation option.

The purpose of index is to associate the original data item with the (jointly)-masked EC points, such that the participant can match its original dataset with the intersection of masked datasets. The values of index are generated by the owner of data so as to identify each data record uniquely. When masking the data from its partner, a participant MUST reserve the index for every record. Section 4.2 gives more discussions on the implementation and maintenance of such indexes. Furthermore, Section 5.6 discusses the risk of side channels raised by indexes and describes a common countermeasure to mitigate such threats.

The content of encoded_point strings are treated differently in round 1 and 2.

* Round 1: The sender of EcdhPsiBatch SHOULD encode EC points to octet strings according to point_octet_format negotiated in the handshake phase. The receiver SHOULD decode the octet strings to EC points accordingly. It MUST check that the EC points are on the curve in order to prevent the so-called small group attacks. After masking the received set of EC points with its private key, the participant SHOULD save the jointly masked points if it is allowed to output the intersection.

* Round 2: The sender SHOULD encode the points with the negotiated point_octet_format, and truncate the string if truncation_option is set. The receiver SHOULD treat the contents as octet strings rather than decode them to EC points, as such strings are enough for computing the intersection. Furthermore, if truncation_option is set, the receiver SHOULD also truncate the jointly masked dataset stored by round 1 with the same truncation method before computing the intersection.

Especially, to convert the original data records to EC points over the curve, the participant SHOULD export ekm from the current TLS session as stated by [RFC8446][RFC9266], and map record data to EC point P with P=hash_to_curve(ekm||data).

4. Implementation Considerations

4.1. The Representation of EC Point

When deciding the point encoding format for the elliptic curves except for curve25519, the participants should consider the aspects of bandwidth, storage and computation comprehensively. Compressed point format could decrease the costs of transmission and storage, but it also needs computation resources to "decompress" the point. Uncompressed format requires more storage spaces and needs more time to transmit, but the participant can perform scalar multiplications without extra effort to recover the points.

For example, when both parties are deployed in the same data center and linked with a high-bandwidth local area network (LAN), they can choose to use the uncompressed format to achieve better performance.

As another example, the parties may be deployed in geographically separated data centers connected with low bandwidth wide area network (WAN), and equipped with high-end computation acceleration devices such as Graphics Processing Units (GPUs). In this case, the computation resource may not be the bottleneck, and the participants can choose the compressed format to minimize the cost of data transmission.

4.2. The Management of Index

A RECOMMENDED method of maintaining indexes is storing the records with a database table and using the row numbers or prime keys as indexes. The intersection result can be matched with original data items by a simple join operation. The participant could also design a different indexing mechanism on its own, as long as the index can be used to identify a record uniquely.

This document uses explicit indexes to identify data items rather than treating the order of records as "implicit" indexes. Compared with the implicit counterpart, explicit indexes can be efficiently created and maintained by modern databases, and do not require the participants to implement extra logic to preserve the order of records.

After masking the EC points from its partner, a participant MUST send the jointly masked data with correct indexes. To achieve this, it keeps the indexes of each data items received in the first round, masks the records with its private key, and associates the masked records with the same indexes. These operations can also be implemented easily with database operations by viewing the received records and masked records as columns in the same table, which enables them to be linked naturally via the prime keys.

4.3. Large Record Size

The EcdhPsiBatch message can be very large. For example, if the dataset contains 2^30 records, the size of encoded EC points included in a single EcdhPsiBatch message can be over 60 GB. Participants of ECDH-PSI SHOULD take great care when implementing the underlying communication channel, and guarantee that there are enough buffer or storage space for sending or receiving such messages.

The methods of sending and handling large messages are beyond the scope of this document.

5. Security Considerations

For PSI protocols, the foremost security goal is ensuring that the private data elements (i.e., records not in the intersection) are not revealed. Neither a protocol partner nor an external attacker can obtain such private elements with the protocol.

This section describes the threat model related to ECDH-PSI, discusses the threats of key reuse, MITM attack, replay attack and side channel, and finally gives probability bounds related to the truncation mechanism.

5.1. Threat Model

Traditionally, ECDH-PSI protocols are considered secure under the so-called semi-honest setting, where both participants will follow the protocol specification, but try to guess each others’ private inputs by observing the received messages.

This document considers an extended setting of malicious attackers. In such setting, ECDH-PSI cannot guarantee that the participants always output correct results, as a malicious participant can send arbitrary messages that produce wrong results. However, it is excepted that ECDH-PSI can somehow preserve the main security goal of data privacy even under such a setting.

In particular, the threat model considers external attackers who have the privilege to access the communication channel between the participants, and internal attackers who act a participant of the protocol.

In this document, most external attackers are excluded by TLS protocol [RFC8446] with mutual authentication, where the participants are authentcated mutually, and the messages are encrypted and authenticated. With the protection of TLS, the external attacker will not be able to inject, replay, truncate or manipulate messages transferred by the channel. Any malicious behavior will be detected immediately by TLS, and the relevant data will be discarded without passing to the application layer. However, TLS alone cannot prevent the Man-In-The-Middle (MITM) attack performed by a malicious node who concurrently runs two sessions and forwards messages from one session to another, which is discussed by Section 5.4.

Malicious internal attackers may send arbitrary messages during the protocol execution. That is to say, it may sends:

* Malformed handshake messages.

* Malformed masked points.

* Malformed jointly masked points.

In fact, malformed handshake message or jointly masked points may eliminate the protocol execution due to incorrect message syntaxes or mislead the victim to output wrong result, which will not harm the major security goal of data privacy. However, the attacker can use malformed masked points to construct a static ECDH oracle, as the victim will always mask those points with its own private key and send the results back. That is to say, if P1 is a malformed masked EC point sent by the attacker and sk is the victim’s secret key used in ECDH-PSI, the attacker will get P1^sk. Once the attacker obtains some advantages with the querying process and calculates sk, it can immediately calculate sk^-1 and use it to "de-mask" any data masked by the victim.

5.2. Static ECDH Oracle

In this section, we analysis the risk of static ECDH oracle as described by Section 5.1, where each ECDH-PSI session can be utilized as an oracle for the ECDH-PSI key used in the current session. The attacker can declare that the size of its dataset is very large, and query the oracle as many times as it could. However, once the session ends, the victim will delete the ECDH-PSI key and use a freshly generated key for new sessions, which means that the attacker

will query different oracle instances in different sessions.

In general, let v be the maximum time of querying an oracle instance, the static ECDH oracle may decrease the security level of elliptic curve by about log(2,v)/2 bits. The problem of static ECDH oracle has been studied extensively in a series of literatures including [BG04][Gra10][MU10][JV13].

For the curves employed by this document, the most efficient attack strategy is still the one described by [BG04]. In particular, let u and v be a pair of integers satisfying u*v=r-1, the [BG04] attack requires the attacker to query the oracle v times and uses

2(sqrt(u)+sqrt(v))=2(sqrt((r-1)/v)+sqrt(v))

scalar multiplications to calculate the secret key. Take the relationship that the security level of an elliptic curve group can be approximately seen as n=log(2,sqrt(r)), the number of scalar multiplications can be seen as 2(2^n/sqrt(v)+sqrt(v)). However, in ECDH-PSI, v is limited by the maximum number of elements holding by one of the participants, which is 2^64 since the parameter of batch_count is carried by an uint64. That is to say, for all the group parameters employed by this document, 2^n/sqrt(v) will be much larger than sqrt(v), and the attacker needs to perform 2^n/(sqrt(v)/2) multiplications, which approximately means the security level will be decreased by log(2,sqrt(v)/2)=log(2,v)/2-1 bits. For the maximum value of v, which is 2^64, the conclusion indicates that the security level will be decreased by 31 bits.

For the curves employed by this document, the decrease of 31 bits is acceptable, as the minimum security level is 128 bit, which will be decreased to 128 - 31 = 97 bits. Currently, there dose not exist any evidence that an elliptic curve with security level of 97 bits is vulnerable to practical attacks. Furthermore, this document does not use elliptic curves defined over extension fields which may be vulnerable to the more efficient attacks proposed by [Gra10] and [JV13].

[MU10] describes another attack by combing static ECDH oracles with the notorious small-group attack. To prevent such attacks, the receiver of a set of EC points MUST checks that every point is on the curve, as specified by Section 3.3.

5.3. Key Reuse

This section discusses the risks of reusing ECDH-PSI key across different sessions. In particular, reusing an ECDH-PSI key may decrease the security level of the elliptic curve based on the discussion presented by Section 5.2, or be utilized by a semi-honest attacker to reveal information related with different protocol runs.

As shown by Section 5.2, the key point of [BG04] attack strategy is the maximum query number for a single oracle instance. Reusing ECDH-PSI key will decrease the security level by much more than 31 bits as the allowed number of queries changes to the sum of queries allowed by all instances using the same key. In the worst case where the sum of queries is 2^(n/2), the security level will be decreased by almost a half.

If an ECDH-PSI key is reused across sessions, an attacker can participate these sessions as partners and obtain some extra information. For instance, the victim V may provide two different sets (denoted by set_0 and set_1) in two different sessions, where the elements are masked by the same secret key sk as set_0^sk and set_1^sk. Then, the attacker who joins both the sessions can calculate intersect(set_0^sk,set_1^sk) and learn the size of intersect(set_0,set_1). Despite that the attacker still cannot recover the original set set_0 or set_1, it indeed learns extra information beyond the original output of PSI even in the semi-honest setting.

5.4. Man-In-The-Middle Attack

MITM attack in ECDH-PSI refers to the case that the adversary acts as an agent who transmits messages between two honest participants. If the attacker transmits messages faithfully, it finally learns the cardinality of the intersection as it obtains the jointly masked datasets of both participants. However, as stated above, the attacker cannot break the privacy of original data items as it cannot obtain the ECDH-PSI keys with such attacks.

In this document, such attacks are avoided by the TLS binding mechanism [RFC9266]. In particular, the particpants use the "tls_exporter" channel binding type to export EKM strings from the TLS sessions, and concatenate the EKM string with the original data item as input to the hash_to_curve function as specified by Section 3.3.

If the malicious adversary M performs MITM attacks against participants A and B, it will establish two different TLS sessions denoted by s_a and s_b. Then, A will export EKM ekm_a from TLS

session s_a, and B will export a different ekm_b from s_b, which means that the same data element data will be mapped to different points on the curve with P_A=hash_to_curve(ekm_a||data) and P_B=hash_to_curve(=ekm_b||data). The PSI protocol will finally fail since the same data item cannot be mapped to the same point on the curve, but the attacker also cannot learn the size of the intersection, which provides more privacy guarantee beyond the secrecy of original data items.

5.5. Replay Attack

In [CY20], Cui and Yu design a new attack for concurrent runs of ECDH-PSI sessions in the malicious setting. In this attack, the attacker establishes two sessions with the same victim who uses two different data sets set_1 and set_2 in different sessions, and finally learns the size of intersect(set_1,set_2).

Let sk_1 and sk_2 denote the ECDH-PSI keys that the victim uses in different sessions, the attacker M performs the attack to V as follows (also refer to Figure 3):

* Step 1: M initiates the first session with V, where V maps all elements of set_1 to points as set pset_1, masks them with sk_1, and sends mset_1=pset_1^sk_1.

* Step 2: M initiates another session with V, and obtains mset_2=pset_2^sk_2 as in Step 1.

* Step 3: The attacker chooses r_1 and r_2, and masks the sets received in Step 1 and 2 with rset_1=mset_1^r_1 and rset_2=mset_2^r2.

* Step 4: M "reflects" the set received in session 1 to V in session 2 by sending rset_1 to V in the second session. Then the victim masks rset_1 with its ECDH-PSI key for session 2 and sends dset_2=rset_1^sk_2 to the attacker.

* Step 5: As in Step 4, the attacker sends rset_2 to the victim in session 1, and receives dset_1=rset_2^sk_1.

* Step 6: Upon receiving dset_1 and dset_2, the attacker de-randomizes the sets with r_2^-1 and r_1^-1 respectively, and obtains:

* Step 7: M calculates the intersection of vset_1 and vset_2, which has the same cardinality with the intersection of set_1 and set_2.

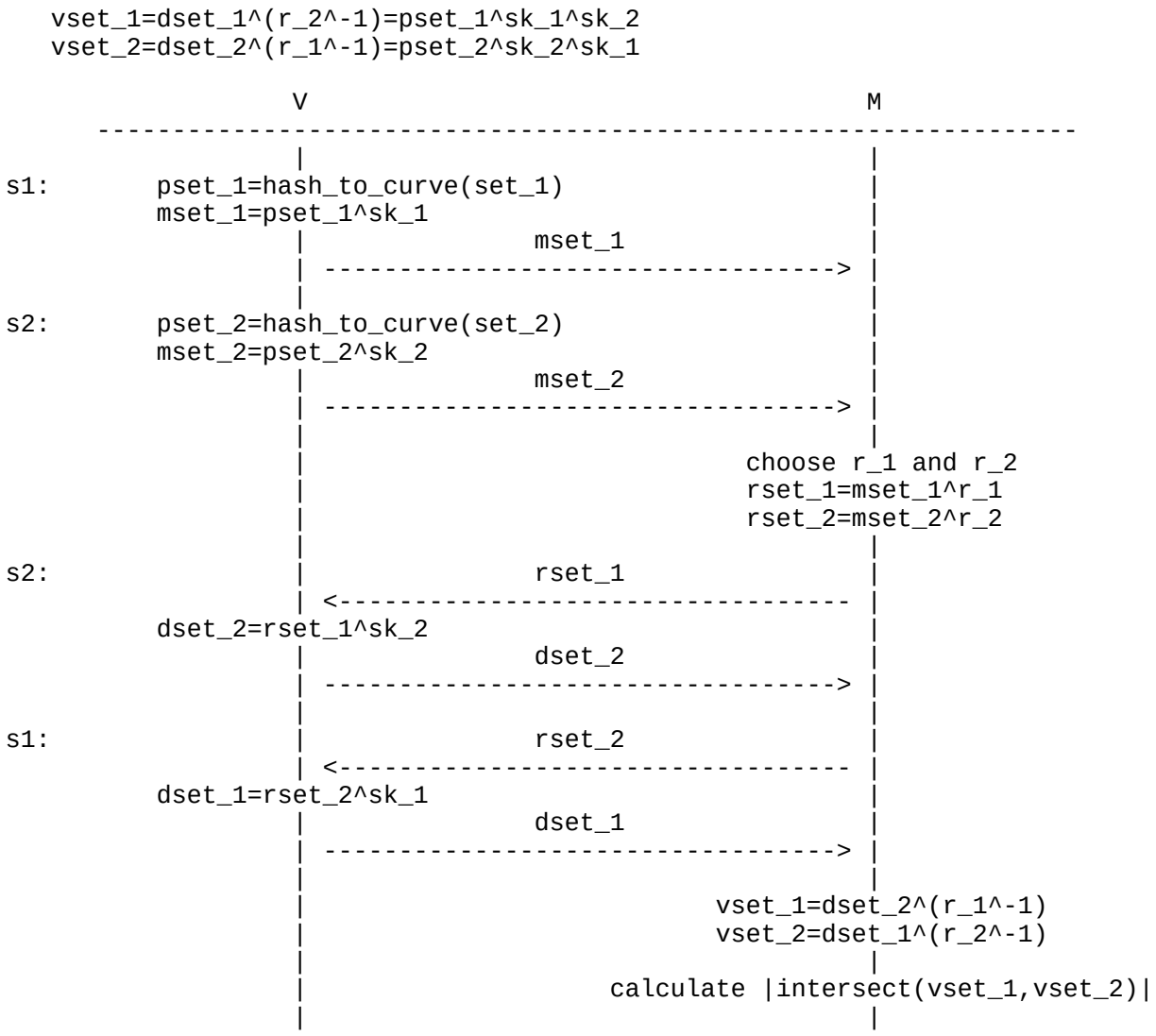

Figure 3: The replay attack against ECDH-PSI

The replay attack can also be mitgated with TLS channel binding mechanism. In particular, if the participants adopt TLS channel binding as specified by Section 3.3, then, in steps 1 and 2 of the replay attack, the victim will extract different EKMs and use them to calculate pset_1 and pset_2 respectively. That is to say, even set_1 and set_2 contain some common elements, they will be mapped to different points in pset_1 and pset_2 as the EKMs are different, which finally fails the attacker in step 7 as overlapped elements are mapped to different points in vset_1 and vset_2.

5.6. Side Channel

In some circumstances, the order of elements and data indexes can be utilized by a semi-honest attacker as a side-channel which leaks information about the original dataset. When two participants A and B execute ECDH-PSI and output the intersection result, one party (say, B) also knows where these common elements appear in A’s list and their indexes, such orders and indexes can be used to infer the orders of inserting those elements to a database table.

A generic method to avoid such side-channels is shuffling the order of elements, as well as the indexes, before they are sent to the partner. However, the shuffling operation may be costly or even practically infeasible when the size of dataset is very large. The participant SHOULD evaluate the risk of side-channels and use suitable mitigation mechanisms when the risk is unacceptable.

5.7. Data Truncation

This section provides a detailed discussion on the truncation mechanism presented in this document.

The truncation, undoubtedly, will raise the probability of message collision. That is, two different data item may be mapped to the same string after the procedures of masking and truncation by coincidence. The collision may happen across the datasets owned by different participants, which causes a false-positive case that two different records are matched by PSI, or happen in the same dataset. The later situation may also lead to a false-positive case. To be more specific, consider a participant A has two different data items data_X and data_Y which are mapped to the same record mask_data_XY. Its partner, say B, also has record data_X which is mapped to mask_data_XY. Finally, A outputs both data_X and data_Y as the result of PSI, as it cannot distinguish which one matches the record of mask_data_XY.

To avoid such false-positive case, we have to consider the collision probability with respect to the sum number of data items owned by A and B. Such probability can be computed with the well-known birthday paradox bound. Let n be the number of sampling and d be the output space, the probability of collision can be approximately computed with:

p(n,d)=1-e^(-n(n-1)/2d)

This document uses two truncation sizes of 128 bit and 192 bit. For 128-bit truncation, the probabilities of different n and d=2^128 are calculated as follows:

* If n=2^50, p(2^50, 2^128) = 2^-29

* If n=2^45, p(2^45, 2^128) = 2^-33

* If n=2^41, p(2^41, 2^128) = 2^-47

* If n=2^40, p(2^40, 2^128) = 2^-49

* If n=2^39, p(2^39, 2^128) = 2^-51

For 192-bit truncation, the probabilities are listed as follows:

* If n=2^50, p(2^50, 2^192) = 2^-93

* If n=2^45, p(2^45, 2^192) = 2^-103

* If n=2^41, p(2^41, 2^192) = 2^-111

* If n=2^40, p(2^40, 2^192) = 2^-113

* If n=2^39, p(2^39, 2^192) = 2^-115

That is, if the number of records is less than 2^40, the probability of false-positive will be smaller than 2^-48 for truncation length of 128 bit, and smaller than 2^-112 for 192 bit. The participant can also decide the truncation option by calculating the collision probability, and only use truncation when they both agree that the probability is acceptable.

6. IANA Considerations

This document may need IANA to allocate hash_to_curve identifiers which may also be used in other applications.

7. References

7.1. Normative References

[ECDSA] American National Standards Institute, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI ANS X9.62-2005, November 2005.

[FIPS186-4] National Institute of Standards and Technology (NIST), "Digital Signature Standard (DSS)", FIPS 186-4, DOI 10.6028/NIST.FIPS.186-4, July 2013, <https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf>.

[GBT.32905-2016] Standardization Administration of China, "Information security technology --- SM3 cryptographic hash algorithm", GB/T 32905-2016, March 2017, <http://www.gmbz.org.cn/upload/2018-07-24/1532401392982079739.pdf>.

[GBT.32918.5-2017] Standardization Administration of the People’s Republic of China, "Information security technology --- Public key cryptographic algorithm SM2 based on elliptic curves --- Part 5: Parameter definition", GB/T 32918.5-2017, December 2017, <http://www.gmbz.org.cn/upload/2018-07-24/1532401863206085511.pdf>.

[RFC2104] Krawczyk, H., Bellare, M., and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, DOI 10.17487/RFC2104, February 1997, <https://www.rfc-editor.org/info/rfc2104>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <https://www.rfc-editor.org/info/rfc5869>.

[RFC7748] Langley, A., Hamburg, M., and S. Turner, "Elliptic Curves for Security", RFC 7748, DOI 10.17487/RFC7748, January 2016, <https://www.rfc-editor.org/info/rfc7748>.

[RFC8174] Leiba, B., "Ambiguity of Uppercase vs Lowercase in RFC 2119 Key Words", BCP 14, RFC 8174, DOI 10.17487/RFC8174, May 2017, <https://www.rfc-editor.org/info/rfc8174>.

[RFC9266] Whited, S., "Channel Bindings for TLS 1.3", RFC 9266, DOI 10.17487/RFC9266, July 2022, <https://www.rfc-editor.org/info/rfc9266>.

[RFC9380] Faz-Hernandez, A., Scott, S., Sullivan, N., Wahby, R. S., and C. A. Wood, "Hashing to Elliptic Curves", August 2023.

7.2. Informative References

[BG04] Brown, D. R. L. and R. P. Gallant, "The Static Diffie-Hellman Problem", <https://eprint.iacr.org/2024/306>.

[CHLR18] Chen, H., Huang, Z., Laine, K., and P. Rindal, "Labeled PSI from Fully Homomorphic Encryption with Malicious Security", Proceedings of the 2018 {ACM} {SIGSAC} Conference on Computer and Communications Security, {CCS} 2018, Toronto, ON, Canada, October 15-19, 2018, October 2018, <https://doi.org/10.1145/3243734.3243836>.

[CY20] Cui, H. and Y. Yu, "A Not-SO-Trival Replay Attack Against DH-PSI", <https://eprint.iacr.org/2020/901.pdf>.

[draft-irtf-cfrg-vdaf-12] Barnes, R., Cook, D., Patton, C., and P. Schoppmann, "Verifiable Distributed Aggregation Functions", Work in Progress, Internet-Draft, draft-irtf-cfrg-vdaf-12, 4 October 2024, <https://datatracker.ietf.org/doc/html/draft-irtf-cfrg-vdaf-12>.

[Gra10] Granger, R., "On the Static Diffie-Hellman Problem on Elliptic Curves over Extension Fields", Advances in Cryptology - ASIACRYPT 2010., DOI https://doi.org/10.1007/978-3-642-17373-8_17, <https://doi.org/10.1007/978-3-642-17373-8_17>.

[I-D.ietf-ppm-dap] Geoghegan, T., Patton, C., Pitman, B., Rescorla, E., and C. A. Wood, "Distributed Aggregation Protocol for Privacy Preserving Measurement", Work in Progress, Internet-Draft, draft-ietf-ppm-dap-12, 10 October 2024, <https://datatracker.ietf.org/doc/html/draft-ietf-ppm-dap-12>.

[IMC] Katz, J. and Y. Lindell, "Introduction to Modern Cryptography, Third Edition".

[ISO-SM2] International Organization for Standardization, "IT Security techniques -- Digital signatures with appendix -- Part 3: Discrete logarithm based mechanisms", ISO/IEC 14888-3:2018, November 2018, <https://www.iso.org/standard/76382.html>.

[ISO-SM3] International Organization for Standardization, "IT Security techniques -- Hash-functions -- Part 3: Dedicated hash-functions", ISO/IEC 10118-3:2018, October 2018, <https://www.iso.org/standard/67116.html>.

[JV13] Joux, A. and V. Vitse, "Elliptic Curve Discrete Logarithm Problem over Small Degree Extension Fields - Application to the Static Diffie-Hellman Problem on $E(\mathbb{F}_{q{5}})$", Journal of Cryptology, Volume 26, pages 119143, (2013), DOI https://doi.org/10.1007/s00145-011-9116-z, <https://doi.org/10.1007/s00145-011-9116-z>.

[KKRT16] Kolesnikov, V., Kumaresan, R., Rosulek, M., and N. Trieu, "Efficient Batched Oblivious {PRF} with Applications to Private Set Intersection", Proceedings of the 2016 {ACM} {SIGSAC} Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016, October 2016, <https://doi.org/10.1145/2976749.2978381>.

[LOGJAM] Adrian, D., Bhargavan, K., Durumeric, Z., Gaudry, P., Green, M., Halderman, J. A., Heninger, N., Springall, D., Thom, E., Valenta, L., VanderSloot, B., Wustrow, E., Zanella-Bguelin, S., and P. Zimmermann, "Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice", Proceedings of the 22nd {ACM} {SIGSAC} Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015, October 2015, <https://doi.org/10.1145/2810103.2813707>.

[Meadows86] Meadows, C., "A More Efficient Cryptographic Matchmaking Protocol for Use in the Absence of a Continuously Available Third Party", 1986 IEEE Symposium on Security and Privacy, <https://doi.org/10.1109/SP.1986.10022>.

[MRH04] Maurer, U., Renner, R., and C. Holenstein, "Indifferentiability, Impossibility Results on Reductions, and Applications to the Random Oracle Methodology", In TCC

2004: Theory of Cryptography, pages 21-39, DOI 10.1007/978-3-540-24638-1_2, February 2004, <https://doi.org/10.1007/978-3-540-24638-1_2>.

[MS21] Lauter, K., Kannepalli, S., Laine, K., and R. C. Moreno, "Password Monitor: Safeguarding passwords in Microsoft Edge", <https://www.microsoft.com/en-us/research/blog/password-monitor-safeguarding-passwords-in-microsoft-edge/>.

[MU10] Menezes, A. and B. Ustaoglu, "On reusing ephemeral keys in Diffie-Hellman key agreement protocols", International Journal of Applied Cryptography (IJACT), Vol. 2, No. 2, 2010, DOI https://doi.org/10.1504/IJACT.2010.038308, <https://doi.org/10.1504/IJACT.2010.038308>.

[RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, DOI 10.17487/RFC5056, November 2007, <https://www.rfc-editor.org/info/rfc5056>.

[RFC8031] Nir, Y. and S. Josefsson, "Curve25519 and Curve448 for the Internet Key Exchange Protocol Version 2 (IKEv2) Key Agreement", RFC 8031, DOI 10.17487/RFC8031, December 2016, <https://www.rfc-editor.org/info/rfc8031>.

[RFC8446] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", RFC 8446, DOI 10.17487/RFC8446, August 2018, <https://www.rfc-editor.org/info/rfc8446>.

[RFC8998] Yang, P., "ShangMi (SM) Cipher Suites for TLS 1.3", RFC 8998, DOI 10.17487/RFC8998, March 2021, <https://www.rfc-editor.org/info/rfc8998>.

[RFC9382] Ladd, W., "SPAKE2, a Password-Authenticated Key Exchange", RFC 9382, DOI 10.17487/RFC9382, September 2023, <https://www.rfc-editor.org/info/rfc9382>.

[RFC9497] Davidson, A., Faz-Hernandez, A., Sullivan, N., and C. A. Wood, "Oblivious Pseudorandom Functions (OPRFs) Using Prime-Order Groups", RFC 9497, DOI 10.17487/RFC9497, December 2023, <https://www.rfc-editor.org/info/rfc9497>.

[RR22] Raghuraman, S. and P. Rindal, "Blazing Fast PSI from Improved OKVS and Subfield VOLE", Proceedings of the 2022 {ACM} {SIGSAC} Conference on Computer and Communications Security, {CCS} 2022, Los Angeles, CA, USA, November 7-11, 2022, November 2022, <https://doi.org/10.1145/3548606.3560658>.

Authors' Addresses

Yuchen Wang
Ant Group
Email: tianwu.wyc@antgroup.com

Wenting Chang
Ant Group
Postfach 330440
Beijing
Phone: +49-421-218-63921
Email: bainuan.cwt@antgroup.com

Yufei Lu
Ant Group
Email: yuwen.lyf@antgroup.com

Cheng Hong
Ant Group
Email: vince.hc@alibaba-inc.com

Jin Peng
Ant Group
Email: jim.pj@antgroup.com