Some Theoretical Conditions for Menezes–Qu–Vanstone Key Agreement to Provide Implicit Key Authentication

Daniel R. L. Brown*

May 21, 2014
WORK IN PROGRESS

Abstract

Menezes–Qu–Vanstone key agreement (MQV) is intended to provide implicit key authentication (IKA) and several other security objectives. MQV is approved and specified in five standards.

This report focuses on the IKA of two-pass MQV, without key confirmation. Arguably, implicit key authentication is the most essential security objective in authenticated key agreement. The report examines various necessary or sufficient formal conditions under which MQV may provide IKA.

Incidentally, this report defines, relies on, and inter-relates various conditions on the key deriviation function and Diffie–Hellman groups. While it should be expected that most such definitions and results are already well-known, a reader interested in these topics may be interested in this report as kind a kind of review, even if they have no interest in MQV whatsoever.

Caution: this report is a work in progress. It may contain serious omissions and errors. Readers should verify any proofs before relying upon them, as always, but perhaps more so than is usual.

Contents

*Certicom Research, BlackBerry

1 Introduction

Menezes, Qu and Vanstone introduced a preliminary version of their key agreement scheme in IEEE P363 draft standards. Then, together with Law and Solinas, they [LMQ+98, LMQ+03] introduced the final Menezes–Qu–Vanstone (MQV) key agreement scheme. For completeness, Menezes–Qu–Vanstone is also formally described in §2 of this report.

This report focuses on the two-pass variant of MQV key agreement in which Alice and Bob have pre-authenticated static public keys, and can simultaneously[1] exchange non-authenticated ephemeral public keys.

Law, Menezes, Qu, Solinas and Vanstone [LMQ+98, LMQ+03] listed a set of intended security goals for two-pass Menezes–Qu–Vanstone key agreement, the primary and most crucial of which was *implicit key authentication* (IKA). Formal definitions of implicit key authentication are provided in §4.

Informally, implicit key authentication (IKA) means that an adversary Eve cannot feasibly learn anything about a session key that Alice would agree upon with Bob, provided that Eve is not given invasive access to any of Alice or Bob's private keys (whether ephemeral or static), or any session keys derived from such private keys. Eve is presumed able to substitute her own ephemeral public key in place of Alice's, because the ephemeral public keys are non-authenticated.

One intended benefit of the ephemeral public keys in MQV key agreement is to provide forward secrecy, which is a security objective beyond that of implicit key authentication. Arguably, Menezes–Qu–Vanstone key agreement can be viewed primarily as an efficient way to add forward secrecy to conventional static (authenticated) Diffie–Hellman key agreement. In this context, the question that this report asks is: how does this efficient manner in which MQV adds forward secrecy impact the more basic implicit key authentication security objective? In other words: has any IKA security been sacrificed for the sake of forward secrecy and efficiency?

1.1 Standardization and Deployment of ECMQV

Elliptic curve Menezes–Qu–Vanstone key agreement is specified in five standards:

- IEEE 1363,
- ANSI X9.63,
- NIST SP 800-56A,
- ISO 11770-3[2], and
- SEC 1.

Elliptic curve Menezes–Qu–Vanstone is included, by reference, in some application standards, such as RFC 3278 and RFC 5480. It was temporarily included in, and then removed from, the Suite B set of cryptographic algorithms.[3]

Elliptic curve Menezes–Qu–Vanstone is deployed in BlackBerry smartphones for securely connecting the smartphone to the enterprise network, among other things.

[1] Usually, simultaneity is not a requirement, so the responder can send a key confirmation in the second pass, permitting one-sided explicit key authentication. This report does not formally examine explicit key authentication, and furthermore, takes the view that making the key authentication explicit is a secondary security objective.

[2] Replaced ISO 15946-3

[3] Its removal is probably simply due to lack of a deployment base. Nevertheless, removal naturally raises a concern of a security flaw, which is part of the motivation for this report.

1.2 Cryptanalytic Security Assurances for MQV

The existing primary public[4] argument for the implicit key authentication security of Menezes–Qu–Vanstone key agreement may well be that no publications report to compromise the IKA security of MQV, despite the incentives due to its standardization and deployment. Let us call this the *lack-of-attack* argument.

Two classes of incented attackers are the following.

- A first class of attackers exploit any attacks that they know against deployments, and therefore would be unlikely to publicize their attacks, to avoid their targets taking countermeasures. Exploitative attackers would exploit deployments, not standards. In most cases, such exploitative attackers have many other attack options, such as malware, social engineering, and buffer overflows, meaning that it is plausible, that exploitative attackers would not bother to even to try compromise cryptographic algorithms in the deployment, unless a flaw is obvious and severe. No published evidence supports the existence of an attacker capable of compromising the IKA security of MQV.

- A second class of attackers would seek to publish any attacks that they find, rather than exploit them. They might do so for several reasons: they seek to protect users (and thereby gain some appreciation in various forms, such as future consulting opportunities), they seek to promote their own specific alternative cryptographic schemes, or they are unwilling to undertake the risks involved in exploiting attacks. They might also suspect that exploitative attackers have already discovered the very same attacks, and reasons that most of the gains from exploitation would have been exhausted. These attackers are generally specialized researchers, and focus more on public documents, such as standards, with deployments serving mainly as a bonus incentive.

It is generally uncommon for either of the classes above of attackers to publish a report about having tried but failed to find an attack on a scheme. The standards for MQV, and the original research papers proposing MQV, may construed as reports of such a nature. So, unfortunately, it is not well documented how much effort has gone into trying to find IKA attacks on MQV.

Beyond implicit key authentication, Kaliski found attacks on variants of MQV. Krawczyk [Kra05] found some invasive attacks on variants of MQV. These attakcs are outside the scope of this report, becuase they fall outside of the formal definition of implicit key authentication. Nevertheless, they do show that some effort has gone into cryptanalyzing MQV, and the fact these illustrious researchers did not find IKA attakcs is re-assuring for the IKA security of MQV.

Meanwhile, incentives for attacking Menezes–Qu–Vanstone have been present, because of its standardization and deployment. Naturally, the incentives are smaller than for other more widely used key establishment schemes, such as Rivest–Shamir–Adleman and Diffie–Hellman, but greater than more recent proposals, such as HMQV and the like.

Remark 1.2.1. A counterargument to the lack-of-attack argument is that Menezes–Qu–Vanstone may have actually received inadequate attention, despite its standardization and deployment and the incentives alluded to above. Therefore, Menezes–Qu–Vanstone may be subject to some overlooked defect. This reports strives to examine that possibility, by partially reducing the IKA security Menezes–Qu–Vanstone down to properties of its underlying components.[5] Naturally, the lack-of-attack argument should still be deemed stronger security assurance than any of the results of this report, if only because this report is only the product of a single author.[6] It is not intent of this

[4] Private arguments may have been developed during the design of MQV.

[5] This can be viewed as an afterthought.

[6] Hopefully, this report would provide re-assurance, though the more cynical could construe any such efforts as an ill omen.

report to say that Menezes–Qu–Vanstone has been understudied, or that this report represents a breakthrough in understanding about the security of Menezes–Qu–Vanstone.

1.3 Previous Reductionist Security Results on MQV and its Components

One aim of the research approach used for this report is to avoid making hidden assumptions. Casual reading of previous work[7] might miss some important nuances, thereby introducing hidden assumptions. At some point, previous work must be consulted.

A second aim in this research was to first conduct an independent investigation, to develop a better understanding, and then to compare to past work, mainly in the hope of confirmation. This second aim has not been met satisfactorily in regards to the aspect of improved understanding[8]. The independence aspect has been somewhat well achieved, although the influence of general received wisdom has not been ignored.

Since this is an incomplete work in progress, a detailed comparison to previous work has been deferred. Generally, a prudent reader should presume by default previous work supersedes and subsumes the work in this report.

1.3.1 Key Derivation and Hash Functions

Menezes–Qu–Vanstone key agreement, as conisdered in this report, includes a key derivation function. This is flexible deterministic function, which compress or expand its input, and have public or private inputs. In this report, we assume that the private input is the MQV shared secret value, and attempt to characterize the security requirements therefrom.

One can view the key derivation function as a hash function. Indeed, some MQV standards instantiate the key derivation function using a hash function. Andreeva and Stam [AS11] study keyed hash functions, extending the work of Rogaway and Shrimpton [RS04]. If one reduces the Andreeva–Stam key space to a singleton set, then our opener and inverter definitions seem to correspond to their domain-oriented preimage resistance and range-oriented preimage resistance respectively. I have not yet compared their results to the results in this report. The reader should presume that any correct results in this report are coroloaries of some Andreeva–Stam theorem in [AS11].

Of course, much other previous work has considered the more specific task of key derivation. Krawcyzk considers a more general purpose task and proposes an extract-and-expand key derivation called HKDF, which has since been standardized (by NIST). I have not yet compared the security definitions or results for HKDF to the definitions and results in this report. So, again, until further study, this report's finding should be presumed as corollories of the Krawczyk's findings on HKDF.

1.3.2 General Key Agreement

Much past work has done on defining security for key agreement. Many of these definitions cover a variety of key agreement shapes, such as three or four passes, and cover important secondary security objectives, such as forward secrecy and key-compromise impersonation. By contrast, this report only covers the special case of two-pass key agreement, and only defines implicit key authentication. Accordingly, the security definitions in this report should subsumed as special cases previous, more general, security definitions.

[7] I suspect myself of such dereliction.

[8] At least, I feel I understand key agreement even less than before.

Nevertheless, to make report more self-contained, definitions of implict key authentication are given in this report.

1.3.3 DH-Related Analyses

A tremendous body of previous work exists for Diffie–Hellman key agreement. This report includes portions that are general enough to relate to both Diffie–Hellman and Menezes–Qu–Vanstone key agreement. (Specifically, MQV key agreement uses a conversion function to combine ephemeral and static DH keys; but otherwise relies on the some of the same underlying components as DH key agreement.)

Accordingly, some of the formal results stated in this report also pertain Diffie–Hellman key agreement. The reader should presume that such results, where correct, are equivalent to known results from previous work.

1.3.4 MQV-Related Analyses

Rogaway, Bellare and Boneh [RBB01] provide a security evaluation of ECMQV as specified in SEC 1. They do not provide security proofs, but do note the security of ECMQV relies on certain problems being hard. Some of these problems may be similar to those described in this report (indication in this report of such similarities to be completed).

Yongge Wang [Wan02] established certain security properties of Menezes–Qu–Vanstone in the generic group model.

Krawczyk [Kra05] introduced HMQV, a modification of Menezes–Qu–Vanstone, that aims, firstly, to correct some perceived security flaws in Menezes–Qu–Vanstone, and, secondly, to provide provable security arguments for a key agreement scheme while retaining many of the efficiency advantages of Menezes–Qu–Vanstone. Subsequently, further variations on HMQV were published by other authors.

Kunz-Jacques and Pointcheval [KJP06] sketch a proof of security for Menezes–Qu–Vanstone key agreement.[9] Their security model, while perhaps not being general as other models for key agreement, is more general than the model in this report. They may rely on the random oracle model, while this report does not. They relate the security of MQV to certain problems. The relations and problems may be similar to the those in this report (indication in this report of such similarities to be completed).

Most of the report was produced independently of the past work above, although may still have been subconsiously influenced by the previous work. As far as I know, any overlap with previous work would be due to the intrinsic nature of MQV, which would be an convergence in security analyses.

1.4 Organization of the Remainder of this Report

The subsequent sections of this report are organized as follows:

- Section 2 describes the Menezes–Qu–Vanstone as it will be analyzed in this report, and briefly describes some more specific variants that have been standardized.
- Section 3 describes some formal notation that is used to describe security definitions and results.
- Section 4 provides four formal definitions for implicit key authentication, and describes relations between the definitions, and compares the importance of the four definitions.

[9] Thanks to Berkant Ustaoglu for alerting the author to [KJP06].

- Section 5 defines and relates various conditions on the components of Menezes–Qu–Vanstone key agreement that may be related to implicit key authentication.
- Section 6 proves that certain conditions on the components of Menezes–Qu–Vanstone are necessary for Menezes–Qu–Vanstone to provide implicit key authentication.
- Section 7 proves that certain conditions on the components of Menezes–Qu–Vanstone are sufficient for Menezes–Qu–Vanstone to provide implicit key authentication.

Remark 1.4.1. Scattered throughout the text are remarks such as this. The remarks are meant to be non-essential. So, the main text (exclusive of the remarks) should be self-contained, and not depend critically upon any of these remarks.

Remark 1.4.2. This report has been written in a reference style, with numerous formal results, many of which are independent of each other. Accordingly, it is not strictly necessary to read the report in order, or even the entire report (excluding remarks), to gather some of the main results.[10]

[10] Conjecture: ordered reading suffices for comprehension.

2 Menezes–Qu–Vanstone Key Agreement

Menezes–Qu–Vanstone is a key agreement scheme, in which two parties, say Alice and Bob, can agree upon a secret session key. Each contributes a pre-authenticated public key (by means of a certificate, for example), which will assure each of the other's identity. At the time of the key agreement, Alice and Bob also exchange (unauthenticated) ephemeral public keys, which helps to provide forward secrecy.

2.1 Specifications and Variants of MQV

This report considers only *two-pass* or *full* Menezes–Qu–Vanstone key agreement, without key confirmation. This section specifies, at a high level, this variant of Menezes–Qu–Vanstone key agreement. For greater detail, see §A.1.

Remark 2.1.1. Variable names closely follow Krawczyk's conventions [Kra05]. Additive notation is used for the underlying group, because the efficiencies provided by MQV key agreement are best matched with the efficiencies of using an elliptic curve group, whose group operation is usually written additively.

Remark 2.1.2. The variant of Menezes–Qu–Vanstone described here is slightly more general than in most standards. The intention of this abstraction resulting from this greater generality is to establish the conditions under which Menezes–Qu–Vanstone key agreement is secure. The specializations of Menezes–Qu–Vanstone in standards documents are often noted at appropriate points in this report. The impact upon these specializations upon the conditions are sometimes noted in this report.

2.2 Components

Menezes–Qu–Vanstone key agreement will be regarded in this report as being parameterized by three components:

1. a group $\mathbb{G}$ of prime order n, together with a selected generator G,
2. a conversion function[11] $f : \mathbb{G} \to \mathbb{Z}_n$ where $\mathbb{Z}_n$ is the set of integers modulo n, and
3. a key derivation function $d : \mathbb{G} \to \mathbb{K} \subseteq \{0,1\}^*$.

How these components are used in Menezes–Qu–Vanstone key agreement is described briefly in Figure 1. Greater detail is provided in §A.1.

Remark 2.2.1. This report examines what conditions on these three components are necessary, or sufficient, for Menezes–Qu–Vanstone key agreement to achieve certain security goals. Standards for Menezes–Qu–Vanstone key agreement specify each of these three components. The specific components can then be further examined to see whether they meet the conditions in this report.

Remark 2.2.2. The abstraction of the components may help conduct a security analysis of Menezes–Qu–Vanstone. Certainly, questions about some of the specific components, such as: "What security functions does a key derivation functions built from a hash function like SHA-1 provide?", have already been examined in previous work, perhaps in other contexts. But this report tries to address the question: "What security features are needed of the components by Menezes–Qu–Vanstone?". Abstraction helps to distinguish these two questions. As long as the answers to the first type of question (intrinsic properties of the components) form a subset of the answers to the second type of question (extrinsic requirements for MQV), then conditions required for security should hold.

[11] Also known as *associated value function*.

Alice		Bob
$A = aG$	$\xrightarrow[\text{Authentic}]{A}$	
	$\xleftarrow[\text{Authentic}]{B}$	$B = bG$
$X = xG$	$\xrightarrow[\text{Ephemeral}]{X}$	
	$\xleftarrow[\text{Ephemeral}]{Y}$	$Y = yG$
$k = d((x + f(X)a)(Y + f(Y)B))$		$l = d((y + f(Y)b)(X + f(X)A))$

Figure 1: Menezes–Qu–Vanstone Key Agreement Between Alice and Bob

3 More on Notation

This report adopts some formal notation, described in this section, not too unlike notation used in many earlier cryptology reports of many other authors.

3.1 Probability Distributions

If v is a variable and D is a probability distribution, then, in this report,

$$v \leftarrow D \tag{3.1.1}$$

indicates that v is assigned a value taken from distribution D. Furthermore, this notation, when contained in a larger process, indicates that v is independent of all previously assignments of random variables. Notation $u, v, \ldots, w \leftarrow D$ means the combination of $u \leftarrow D$, $v \leftarrow D$, ... and $w \leftarrow D$. In other words, each of the values $u, v, \ldots, w$ is distributed according to D, and $u, v, \ldots, w$ are independent. Notation $(u, v) \leftarrow D$ means the values of D consist of pairs and are to be parsed accordingly.

Remark 3.1.1. So, $u, v \leftarrow D$ and $(u, v) \leftarrow D$ have entirely different meanings.

If D and E are probability distributions, then, in this report, notation $D \vee E$ indicates the probability distribution that with probability $\frac{1}{2}$ takes a value from D, and otherwise takes a value from E. Thus the probability of a value under distribution $D \vee E$ is the mean of its two probabilities under D and E. (Occurring less often, the similar notation $D \vee_n E$ indicates the distribution with probability $\frac{1}{n}$ takes a value from D, and otherwise, with probability $\frac{n-1}{n}$ takes a value from E.)

Unless otherwise noted, any set $\mathbb{S}$ used in the context of probability distributions indicates the uniform probability distribution on the set $\mathbb{S}$. For example $x, y \leftarrow \mathbb{S}$ means that values x and y are chosen uniformly and independently from set $\mathbb{S}$.

A single value x, used in the context of a probability distribution indicates the distribution that has probability that has probability 1 of taking on value x. This convention is mostly used in combination with the distribution operator $\vee$, as in $v \leftarrow x \vee D$, which means that, with probability $\frac{1}{2}$, v gets assigned value x, and otherwise v is distributed according to D. If D has nonzero probability for x, then the v may also get assigned x from D, so the probability that $v = x$ could be greater than one half.

Any probabilistic algorithm ALG, possibly including its inputs, used in the context of a probability distribution, means that distribution is that of the output of ALG. In other words, algorithms can define distributions. So, $v \leftarrow \mathsf{ALG}(x)$ means that v is produced as the output of ALG with input x.

A notation to define an algorithm ALG, in this report, is

$$\mathsf{ALG}(x, \ldots, y) \rightarrow F(x, \ldots, y) \tag{3.1.2}$$

where $x, \ldots, y$ are the inputs of ALG and F is some formula. More complicated algorithms, such as probabilistic algorithms, may be defined as a larger process, involving assignment of intermediate variables. By a slight abuse of notation, the input variables may appear above a line of the form (3.1.2).

To use the output of ALG in a formula, we may write $\mathsf{ALG}(x, \ldots, y)$, effectively, as a shortcut to assigning an intermediate value $v \leftarrow \mathsf{ALG}(x, \ldots, y)$. A formula F in the definition of an algorithm such as (3.1.2) may use the output of other algorithms.

Remark 3.1.2. The defining arrow $\rightarrow$ is the opposite direction of the variable getting assigned as the output of an algorithm. In both cases, the value at the head of the arrow is the output of the algorithm, and the algorithm with its inputs is at the tail of the arrow.

3.2 Some Operators

Let $\exp_G : \mathbb{Z}_n \to \mathbb{G}$ be defined by

$$\exp_G(a) = aG. \tag{3.2.1}$$

Function is $\exp_G$ is an isomorphism from $\mathbb{Z}_n$ to $\mathbb{G}$. In this report, this operation is called scalar multiplication. Over groups using multiplicative notation, this operation would be called exponentiation. The notation $\exp_G$ is due to the terminology for multiplicatively notated groups.

Remark 3.2.1. Function $\exp_G$ can be evaluated in at most $2\log_2(n)$ group operations, using a double-and-algorithm, which makes an efficiently evaluatable function.

Let $\log_G : \mathbb{G} \to \mathbb{Z}_n$ be defined by the equation

$$\log_G(A)G = A. \tag{3.2.2}$$

Then $\log_G$ is an isomorphism from the group $\mathbb{G}$ to the additive group of $\mathbb{Z}_n$. It is the inverse of the isomorphism $\exp_G : \mathbb{Z}_n \to \mathbb{G} : a \mapsto aG$.

Remark 3.2.2. Conjecture 5.2.2 says that function $\log_G$ is not efficiently computable over elliptic curve groups, unlike $\exp_G$ (Remark 3.2.1).

Let

$$A \otimes B = abG \tag{3.2.3}$$

where $A, B \in \mathbb{G}$ are such that $A = aG$ and $B = bG$, for some $a, b \in \mathbb{Z}_n$. Equivalently, $A \otimes B = \log_G(A)B$. The operator $\otimes : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ is a well-defined function because $A \otimes B$ is uniquely determined.

Remark 3.2.3. Similarly to remark 3.2.2 about $\log_G$, binary operation $\otimes$ appears hard to compute.

Let

$$\mathsf{mqv}(A, B, X, Y) = d((X + f(X)A) \otimes (Y + f(Y)B)), \tag{3.2.4}$$

for $A, B, X, Y \in \mathbb{G}$ and $f : \mathbb{G} \to \mathbb{Z}_n$ and $d : \mathbb{G} \to \mathbb{K}$. As with the operator $\otimes$, the value $\mathsf{mqv}(A, B, X, Y)$ is unique, so mqv is a well-defined function.

Remark 3.2.4. One of the questions central to this report, because of its relation to the capacity of Menezes–Qu–Vanstone providing implicit key authentication, is whether the function mqv is hard to compute.

Let

$$a \mathrel{\mathring{=}} b \tag{3.2.5}$$

be 1 if a and b have the same value, and 0 otherwise. In other words,

$$(a \mathrel{\mathring{=}} b) = \begin{cases} 1 & \text{if } a = b \\ 0 & \text{if } a \neq b. \end{cases} \tag{3.2.6}$$

Remark 3.2.5. This is a ubiquitous operator.

Remark 3.2.6. In computer language C, this operator is written `==`. In computer language J, this operator is written `=`.

Remark 3.2.7. In some fields of mathematics, the value of $(a \mathrel{\mathring{=}} b)$ is expressed by $\delta(a - b)$ where δ is known as the Kronecker delta function, which is 1 at 0, and 0 elsewhere.

Remark 3.2.8. Yet another notation, used in the book Concrete Mathematics, for this binary operation is $[a = b]$.

Remark 3.2.9. Much other work in cryptology does not resort to this operator. Instead, it uses the probability notation $\Pr[a = b]$.

This implicitly defines the symbol $=$ to be an operator whose output is an event in a probability space.

Remark 3.2.10. The expected value of $a \mathrel{\hat{=}} b$ is $\Pr[a = b]$.

Remark 3.2.11. This report generally define success rates as the expected of a binary-valued success indicator. Sometimes this success indicator is the output of the operator $\mathrel{\hat{=}}$, but in other cases, it is the output of the next operator below.

Remark 3.2.12. A mnemonic for the symbol $\mathrel{\hat{=}}$ is as $1|0$ rotated by a counter-clockwise quarter-turn, with the 1 landing inside the equal sign, and the zero outside the equal sign.

Let $a \mathrel{\hat{\sim}} b$ be defined by

$$(a \mathrel{\hat{\sim}} b) = -(-1)^{a \mathrel{\hat{=}} b}. \tag{3.2.7}$$

Equivalently, $(a \mathrel{\hat{\sim}} b) = (2(a \mathrel{\hat{=}} b) - 1)$ and

$$(a \mathrel{\hat{\sim}} b) = \begin{cases} 1 & \text{if } a = b \\ -1 & \text{if } a \neq b. \end{cases} \tag{3.2.8}$$

Remark 3.2.13. A mnemonic for the symbol $\mathrel{\hat{\sim}}$ is to go up or down, depending on the equality, or when the symbol is turned sideways, as have a value either side of zero.

Remark 3.2.14. In this report, the operator $\mathrel{\hat{\sim}}$ is usually applied to arguments in the set $\{0, 1\}$, in other words, to single bits, or to boolean values. Often one of the bits is the output of an application of the $\mathrel{\hat{=}}$ operator.

Remark 3.2.15. The operator $\mathrel{\hat{\sim}}$ is generally used in, and only in, a random process that involves a split of distributions, such as $D \vee E$.

Remark 3.2.16. As will be seen in subsequent sections, taking the expected value of the $\mathrel{\hat{=}}$ and $\mathrel{\hat{\sim}}$ operators provides a consistent normalization of success rates across computational and decisional algorithms, such that the reductions between them preserves success rates.

3.3 Cost of Algorithms

The computational cost of an algorithm ALG will be indicated as $\gamma(\mathsf{ALG})$. Detailed units of cost are treated heuristically. Approximately, they could consist of an integer corresponding to the number of bit operations. Computational cost could also consist of a vector, with its components measuring differing aspects of computational cost, such as bit operations and memory.

The cost of implementing functions f and d, components of Menezes–Qu–Vanstone key agreement, are indicated $\gamma(f)$ and $\gamma(d)$, respectively. Generally, these costs are considered to be small from an adversary's perspective. For the group operation in $\mathbb{G}$, we write $\gamma(\mathbb{G})$ for its cost. The cost of scalar multiplication in the $\mathbb{G}$ depends on $\gamma(\mathbb{G})$, the size of the scalar multiplier, and the choice of algorithm. However, the cost of a scalar multiplication is not generally much larger than $2 \log_2(n) \gamma(\mathbb{G})$.

Notation $\gamma(\$)$ is used for the cost of generating a random bit. For simplicity we will approximate the cost of generating a random integer z in $\mathbb{Z}_n$ as at most $2 \log_2(n) \gamma(\$)$.

More generally, the focus of this report is on security, not efficient implementation. Firstly, minor efficiency differences in adversaries do not make a major difference in security, so the cost of the algorithms are not treated very precisely in this report. Secondly, many results in this report are of a form where one algorithm is shown to have a cost of at most another algorithm of unknown cost plus some known costs. It is usually a reasonable conjecture that the unknown cost is much larger than the known costs. So the known costs are not treated very precisely.

In particular, arithmetic operations in the ring $\mathbb{Z}_n$ and simple operations on bit strings and group elements, such as comparisons, will be ignored in the cost assessment of algorithms, on the grounds that these operations have a small cost, and are usually dominated by costs of other hypothetical algorithms for problems believed to be hard.

3.4 Problems, Games, Experiments and Implications

This report will generally describe an adversary or a solver of a problem, in terms of a process[12] under which an algorithm, or a suite of algorithms, is measured. The algorithm must have inputs and outputs of a certain data types. The success rate of the algorithm depends on the outcome of the process as it interacts with the algorithm.

As a matter of notational convenience, the variable label used to represent an adversary or solver a problem, will be derived from the description of the whole class of adversary or problem.

The success rate of an algorithm A will be indicated as $\sigma(\mathsf{A})$, provided that it is clear, such as from the label given to the algorithm, what problem that the algorithm is trying to solve.[13]

If each of P and Q is an adversary, or a solver of a problem, the notation $\mathsf{P} \Rightarrow \mathsf{Q}$ may informally summarize a formal result that the algorithm for Q may be constructed from an algorithm for P, with little increase in computational cost and little decrease in success rate. The informal notation $\mathsf{P\&Q}$ is used when both P and Q are hypothesized, or concluded, to exist, or to be constructible, with both with some reasonable or relative success rate and cost, in summarizing a formal result. The informal notation $\mathsf{P|Q}$ is used when P and Q are hypothesized, or concluded, to exist, or to be constructible, with reasonable or relative computational cost, and reasonable or relative total success rate.

Remark 3.4.1. Typically, $\Rightarrow$ is a transitive relation, so if $\mathsf{P} \Rightarrow \mathsf{Q}$ and $\mathsf{Q} \Rightarrow \mathsf{R}$, then $\mathsf{P} \Rightarrow \mathsf{R}$.

Remark 3.4.2. The transitivity of $\Rightarrow$, as in Remark 3.4.1, sometimes breaks down. It is conjecture in this report that this break-down occurs only when some of problems being related are *quizzical* problems, which are oracle-assisted problems such that oracles have conditionally bounded success rates, and the bounded success rates of the oracles for the quizzical problems are related to the success rates of the other problems being related by the $\Rightarrow$ operator at hand.

The informal notation $(\mathsf{P} \Rightarrow \mathsf{Q})$, differing from $algoP \Rightarrow \mathsf{Q}$ only by the addition of parentheses, is used to summarize a new class of adverasry or problem solver, say R, whose essence is to perform that task of Q together with an oracle that completes that task of P. Such problems are often called oracle-assisted problems.

[12] A more specific term, that others have used, is a game or experiment.

[13] A heavier, but arguably more precise, notation would place a subscript under the success rate to indicate the type of process under which success is defined. In the notation of this report, the process for a given algorithm will be clear from context, usually by reference to both the algorithm's description and label.

3.5 Existentiality and Constructibility

Beyond computational cost and success rate, a third dimension to the hardness of a problem, or the security against an adversary, is constructibility. An algorithm to solve a problem with high-success and low computational cost may clearly exist, yet all known constructions may have low success rate or high computational cost. For example, finding collisions in a hash function may be existentially easy, but constructibly hard. If a security criterion of a scheme relies on the collision resistance of a hash function, then we may say that, under the criterion, the scheme is existentially insecure but constructibly secure.

3.6 Formality Versus Essence

The formal notation provides a uniform and consistent way of stating the variety of definitions, and stating and proving a variety of results involving these definitions. Certainly, in some cases, the results are so obvious and familiar[14] that a proof using the formal notation looks wasteful and unnatural. But other results are less familiar[15], at least were so to the author, so the formal notation may help navigate unfamiliar notions[16].

Remark 3.6.1. Although the formality of this notation arguably creates clutter merely to express the simplest ideas, written formality may also help those who, like the author easily muddle and misremember ideas.

Formality may also help one to see past one's preconceptions of the components, say of the key derivation function, and to help one focus on a specific security property, such as implicit key authentication, of which one might otherwise lose sight while examining other security properties, such as forward secrecy.

3.7 Precision of Success Rates

The formal results in this report sometimes contain statements about success rates which include factors which seem to have no practical impact, such as a factor of $(1 - \frac{1}{n})$ or $(1 - \frac{1}{|\mathbb{K}|})$, where n or $|\mathbb{K}|$ must usually be very large, in the success rate. A purpose of striving for such levels of precision is not precision *per se*, but rather an attempt to have rigorous proofs with exact accounting of success rate. By contrast, computational cost was treated loosely, so more explanation is warranted.

This report deals with general security conditions of the components of Menezes–Qu–Vanstone. This report does not examine any specific set of components, but rather what properties the components need to possess to make Menezes–Qu–Vanstone secure. In other words, this report tries to determine the tasks that components must fulfill. Standards provide specific components that serve as the tools to complete these tasks.

A side effect of the generality of the conditions is that degenerate components, that nobody would actually ever propose to use, can be tested against these conditions. A dubious application of this would be to instantiate Menezes–Qu–Vanstone with components that are minimal in the sense that they meet the conditions of this report, but in all other aspects appear defective. For example, one might instantiate Menezes–Qu–Vanstone with a key derivation function in which collisions are easy to find, perhaps with the justification that this collision-ridden key derivation function is faster than the conventional key derivation functions.

Such dubious application is not the intent of this report. The generality of the analysis is not recommendation to throw caution to the wind. Indeed, as noted earlier, this report focuses only one type

[14] Such as MQV relying on the hardness of the discrete logarithm problem.

[15] Such as MQV relying the hardness of prediction problems for the conversion function and key derivation function

[16] Once familiarity has been acquired, in hindsight, the simpler results may appear to be as obvious as already familiar results.

of security, ignoring many secondary security objectives. The secondary security objectives may require further security conditions that have not been identified in this report. One should continue to use the standardized version of Menezes–Qu–Vanstone.

A slightly less tenuous application of the generality of the conditions is that perhaps it will identify some specific security conditions on the components that have been previously overlooked. For example, the key derivation functions currently standardized versions of Menezes–Qu–Vanstone are built from hash functions, such as SHA-1. The first standards for SHA-1 were in the context of computing a message digest in a digital signature. Strictly speaking, the standards did not promise any kind of pseudorandomness properties for the hash functions, and the digital signature algorithms do not seem to require it (although it may be sufficient condition for some proofs). Of course, many subsequent schemes have gone on to rely on pseudorandomness properties of hash functions, so Menezes–Qu–Vanstone is on that same bandwagon. The only point here is that perhaps some quirky unknown conditions may be uncovered.

Perhaps the most useful benefit of the generality of the security conditions is the testing the correctness of the statements. For example, one might substitute degenerate components to test the correctness. This is the main reason that the result statements contain a high level of precision. The reader can test the result statements by applying a degenerate component, without having to first read the proof.

In other words, this report's objective is that the statements aim to withstand reduction to absurdity. This is not to say that less stringent statements which include approximations, cover the typical instantiations, but perhaps fail under degenerate instantiations are not meaningful. It is just for those[17] that do not have sufficient intuition to resolve which aspects of a result statement are critical and which can be safely ignored.

As an example of the pitfalls, consider that a good intuition for standardized component such as a key derivation function built from SHA-1 suggests that its output is independent of the elliptic curve operations. The pitfall here is that because of this intuition, an assumption that this independence is true may be used, but unstated, in the proof. This is unsatisfactory only in the sense that something is hidden in the proof. It would be satisfactory to state precisely the independence condition, and then include it in the statement of the result. It is unsatisfactory if the statement result does not include this condition, but the proof, and perhaps even result statement, does indeed rely upon it.

One way to detect missing assumptions in result statements is to substitute degenerate components. For example, consider the constant key derivation function. Of course, nobody would use such an absurd key derivation function, but the correctness of the result statements with a constant key derivation may serve to identify a property that one normally automatically assume about key derivation functions: that no single value is particularly probable. For another example, consider key derivation that solves the discrete logarithm problem in the elliptic curve group being used. Of course, such a key derivation function is believed to be infeasible to compute. Nevertheless, such a hypothetical degenerate key derivation function may serve to severe our reliance on hidden assumption such as independence of the key derivation function and the elliptic curve group. (Again, such independence may be quite reasonable, but it should not relied upon in a hidden manner.)

Another potential pitfall is the *negligibility pitfall*. Certain events and values of a cryptographic scheme will occur negligibly often when the honest entities are using the key agreement scheme. An implementer of the scheme is likely to consider only what users of the cryptographic scheme would do, and thereby conclude that such event and values would occur negligibly often, and thus be inclined to ignore them. Unfortunately, adversaries may be able to cause negligibly-often-occurring events or values to be used in the cryptographic scheme. If the implementer has not checked for these events of values, then the adversary

[17] Like me

may be able to harm.

The negligibility pitfall can also arise in security proofs if certain special cases are ignored. This is especially exacerbated in complicated proofs involving several oracles, some of which may be provided by the adversary, and some provided to the adversary.

It is hoped that a sufficiently high level of precision in the proofs will catch such pitfalls. Ideally, a proof would apply high precision where, and only where, it is truly important, but when it is not a prior known where the precision is truly important, an approach, the one taken in this report, is to apply high precision uniformly.

The minor factors, such as $(1 - \frac{1}{n})$, that appear in the result statements certainly create clutter for the casual reader. As noted above, these details have been kept in the result statements for checking purposes. Those readers interested in relying upon the results, but not at all interested in checking the results against reductions to absurdity, let alone in checking the proofs, must at minimum inspect the result statements sufficiently enough to extract the parts pertinent to their purpose.

4 Implicit Key Authentication

This section states some formal definitions for implicit key authentication. Four different *grades* of implicit key authentication are considered, each having its own formal definition. Some relations between the four grades are proven.

The formal definitions strictly only apply to the Menezes–Qu–Vanstone key agreement scheme—not to any other key agreement schemes. This scheme-specific approach has the convenience of not having to define a general class of key agreement scheme, of which Menezes–Qu–Vanstone would be a member. The scheme-specific approach has the disadvantage that the security definitions are not automatically generalizable to other key agreement schemes, making it difficult to compare security of Menezes–Qu–Vanstone against other key agreement schemes.

Nonetheless, any two-pass key agreement scheme permitting the pre-arranged exchange of public authenticated values and real-time exchange of public unauthenticated values is sufficiently similar to Menezes–Qu–Vanstone to warrant a comparison. The formal definitions are therefore written to be easily adaptable to such other key agreement schemes by appropriate re-interpretation of the variables A, B, X, Y, and mqv.

4.1 Computational Passive IKA

Definition 4.1.1. *An algorithm* CPIKA *is a computational passive implicit key authentication adversary if it takes as input four elements of* $\mathbb{G}$ *and outputs an element of the session key space* $\mathbb{K}$*. Its success rate* $\sigma(\mathsf{CPIKA})$ *is defined as the expected value of* s *in the following process*

$$A, B, X, Y \leftarrow \mathbb{G} \tag{4.1.1}$$

$$k \leftarrow \mathsf{CPIKA}(A, B, X, Y) \tag{4.1.2}$$

$$s \leftarrow k \stackrel{?}{=} \mathsf{mqv}(A, B, X, Y) \tag{4.1.3}$$

The computational passive implicit key authentication adversary models an adversary Eve who witnesses all the communications between Alice and Bob. Eve succeeds if she computes Alice and Bob's Menezes–Qu–Vanstone session key.

Menezes–Qu–Vanstone key agreement provides computational passive implicit key authentication if no high-success, low-cost adversary CPIKA as in Definition 4.1.1 is constructible.

Remark 4.1.1. The MQV key agreement is essentially symmetric with respect to the parties Alice and Bob (except perhaps with respect to the individualization of the key derivation function d). So, it suffices to define IKA adversaries with respect to Alice.

Remark 4.1.2. The success rate $\sigma(\mathsf{CPIKA})$ is the also probability that $k = \mathsf{mqv}(A, B, X, Y)$.

Remark 4.1.3. It is possible that the success rate of the algorithm CPIKA may be constant over the inputs A, B, X, Y. Consequently, the probability of success is only over the random choices, if any, made by CPIKA. In this case, one can run CPIKA multiple times, say m times, to get another algorithm CPIKA^m. No *a priori* means to select which one of the m computed keys should be the output of CPIKA^m helps. In other words, guessing one the keys at random will not increase the success rate.

Nevertheless, in most cases, Eve may also witness the use of the session key k, such as its use to encrypt and authenticate some data. In this case, Eve can be assume to be able to verify whether any candidate key is indeed Alice and Bob's MQV session key. With this additional information, the algorithm CPIKA^m can find the correct k if it is among the set of m produced by the m runs of CPIKA. So the cost of the repeated algorithm is $\gamma(\mathsf{CPIKA}^m) = m\gamma(\mathsf{CPIKA})$ and the success rate is $\sigma(\mathsf{CPIKA}^m) = 1 - (1 - \sigma(\mathsf{CPIKA}))^m$.

For small enough values of $\sigma(\mathsf{CPIKA})$, the approximation $\sigma(\mathsf{CPIKA}^m) \approx m\sigma(\mathsf{CPIKA})$ is fairly accurate. This suggests that the ratio $\gamma(\mathsf{CPIKA})/\sigma(\mathsf{CPIKA})$ is a good overall measure, given all the conditions above, of the effectiveness of the computational passive implicit key authentication adversary.

The transformation CPIKA to CPIKA^m only works for positive integers m. In particular, the general transformation does not seem to go the other way, so one cannot decrease the computational cost by a factor m in exchange for increasing the success rate. Because of the unidirectional nature of the transformation from CPIKA to CPIKA^m, there is not really full equivalence in trading off success rate for cost. So, there are still effectively two parameters to measure an adversary, which could be taken to be $\gamma(\mathsf{CPIKA})/\sigma(\mathsf{CPIKA})$ and $\gamma(\mathsf{CPIKA})$, where $\gamma(\mathsf{CPIKA})$ is taken to be the least cost in the spectrum of algorithms CPIKA^m.

Such trade-off considerations may apply to other adversaries, and to solvers of problems, with some noted exceptions.

Remark 4.1.4. The repeated-runs approach of Remark 4.1.3 seems to be applicable to a wider class of CPIKA algorithms, as follows. If the success rate $\sigma(\mathsf{CPIKA})$ depends only on its inputs A, B, X, Y—for example, if CPIKA is deterministic—then the previous repeated-runs does not directly apply, but one can instead invoke a kind of random self-reducibility to apply a repeated-runs approach. The repeated-run algorithm CPIKA^m runs CPIKA with modified inputs. Each run selects random integers i and j such that $ij \equiv 1 \bmod n$, and feeds input (iA, jB, iX, jY) to the CPIKA algorithm. This may perhaps cause each run to have the same success rate, and more importantly, cause the runs to have independent probability of success. If so, the same kind of considerations to those in Remark 4.1.3 may apply.

4.2 Computational Active IKA

Definition 4.2.1. *An algorithm* CAIKA *is a computational active implicit key authentication adversary, if it takes as input three elements of* $\mathbb{G}$ *and outputs an element of* $\mathbb{G}$ *and an element of the session key space* $\mathbb{K}$*. Its success rate* $\sigma(\mathsf{CAIKA})$ *is defined as the expected value of* s *in the following process*

$$A, B, X \leftarrow \mathbb{G} \tag{4.2.1}$$

$$(Y, k) \leftarrow \mathsf{CAIKA}(A, B, X) \tag{4.2.2}$$

$$s \leftarrow k \stackrel{?}{=} \mathsf{mqv}(A, B, X, Y) \tag{4.2.3}$$

A computational active implicit key authentication adversary models an adversary Eve who replaces Bob's ephemeral public key with one of her own choosing. Eve succeeds if she can compute the key that Alice would then compute with this modification of Bob's ephemeral public key.

Menezes–Qu–Vanstone key agreement provides computational active implicit key authentication if no high-success, low-cost adversary CAIKA as in Definition 4.2.1 is constructible.

Remark 4.2.1. Unlike a computational passive implicit key authentication adversary, this is an online adversary. Therefore, repeated runs of CAIKA like those discussed in Remark 4.1.3 have an extra cost. Each run requires Alice to use the key k under the modified run, unless the CAIKA computes the same Y every run.

4.3 Decisional Passive IKA

Definition 4.3.1. *An algorithm* DPIKA *is a decisional passive implicit key authentication adversary, if it takes as input four elements of* $\mathbb{G}$ *and an element from the session key space* $\mathbb{K}$*, and yields output that is*

a bit. Its success rate $\sigma(\mathsf{DPIKA})$ *is defined as the expected value of* s *in the following process*

$$A, B, X, Y \leftarrow \mathbb{G} \tag{4.3.1}$$
$$k \leftarrow \mathbb{K} \vee \mathsf{mqv}(A, B, X, Y) \tag{4.3.2}$$
$$b \leftarrow \mathsf{DPIKA}(A, B, X, Y, k) \tag{4.3.3}$$
$$s \leftarrow b \mathrel{\hat{=}} (k \stackrel{?}{=} \mathsf{mqv}(A, B, X, Y)) \tag{4.3.4}$$

A decisional passive implicit key authentication adversary models an adversary Eve who witnesses all communication between Alice and Bob. Eve is successful if she is able to learn anything about the Menezes–Qu–Vanstone session key that Alice and Bob compute.

Menezes–Qu–Vanstone key agreement provides decisional passive implicit key authentication if no high-success, low-cost adversary DPIKA as in Definition 4.3.1 is constructible.

Remark 4.3.1. A DPIKA with success rate $\sigma(\mathsf{DPIKA}) = 0$ is obtained by choosing bit b uniformly at random, simply ignoring all its inputs. This is a low-cost DPIKA, with $\gamma(\mathsf{DPIKA}) = \gamma(\$)$.

Remark 4.3.2. Another low-cost DPIKA, with slightly higher success rate, can be constructed by emulating the fact that Definition 4.3.1 selects $k = \mathsf{mqv}(A, B, X, Y)$ with probability $p = \frac{1}{2} + \frac{1}{2|\mathbb{K}|}$, by selecting $b = 1$ with this probability and $b = 0$ otherwise, again ignoring all its inputs. The success of this algorithm is $\sigma(\mathsf{DPIKA}) = (p - (1-p))^2 = \frac{1}{|\mathbb{K}|^2}$. This algorithm can be implemented with computational cost of roughly $(1 + \log_2(|\mathbb{K}|))\gamma(\$)$.

Remark 4.3.3. A yet more successful low-cost DPIKA, but still with a low success rate for large $|\mathbb{K}|$ can constructed by fixing $b = 1$, taking advantage of the fact that probability in the previous remark is greater than $\frac{1}{2}$. The success rate of this algorithm is $\frac{1}{|\mathbb{K}|}$.

Remark 4.3.4. If DPIKA is decisional passive implicit key authentication adversary, then the algorithm DPIKA^- defined by $\mathsf{DPIKA}^-(A, B, X, Y, k) \to 1 - \mathsf{DPIKA}(A, B, X, Y, k)$ is also a decisional passive implicit key authentication adversary, with cost $\gamma(\mathsf{DPIKA}^-) = \gamma(\mathsf{DPIKA})$ and success rate $\sigma(\mathsf{DPIKA}^-) = -\sigma(\mathsf{DPIKA})$.

More generally, any decisional adversary, or problem solver, with a negative success rate can be converted into one with a positive success rate of the same magnitude.

Remark 4.3.5. The success rate of DPIKA is almost equal to the following difference of probabilities. Consider two processes:

$$A, B, X, Y \leftarrow \mathbb{G} \tag{4.3.5}$$
$$k \leftarrow \mathbb{K} \tag{4.3.6}$$
$$b_0 \leftarrow \mathsf{DPIKA}(A, B, X, Y, k) \tag{4.3.7}$$

and

$$A, B, X, Y \leftarrow \mathbb{G} \tag{4.3.8}$$
$$k \leftarrow \mathsf{mqv}(A, B, X, Y) \tag{4.3.9}$$
$$b_1 \leftarrow \mathsf{DPIKA}(A, B, X, Y, k) \tag{4.3.10}$$

The the success rate $\sigma(\mathsf{DPIKA})$ is approximately the probability p_1 that $b_1 = 1$ minus the probability p_0 that $b_0 = 1$. Precisely,

$$\sigma(\mathsf{DPIKA}) = p_1 - p_0 + \frac{1}{|\mathbb{K}|}(2p_1 - 1). \tag{4.3.11}$$

Such differences of probabilities, as in $p_1 - p_0$, (or, sometimes even divided by two!) are usually called the *advantage* of a distinguisher, and commonly used to measure decisional problems and adversaries.

The formalized success rate of decisional algorithm used in this report is definitionally closer to the success rate of computational algorithm than the latter advantage (difference of probabilities above) is, because both success rates, computational and decisional, are defined as the expected value of some success indicator in both the computational case and decisional case, whereas the advantage is defined in terms of two games, and not as an expectation.

Also, in theory, the best possible DPIKA that could exist is the one that somehow computes $k \triangleq \mathsf{mqv}(A, B, X, Y)$. Under the definition of this report, such an algorithm would have success rate one. Its advantage, however, would only be $1 - \frac{1}{|\mathbb{K}|}$. Indeed, the highest possible advantage is $1 - \frac{1}{|\mathbb{K}|}$.

Yet another reason for considering success rate rather than advantage is the following. For smaller values of $|\mathbb{K}|$, which can arise in one-time pads applied to short messages, success rate and advantage can differ by larger amounts. In the extreme $|\mathbb{K}| = 1$, it is easy to find a success rate 1 adversary, but any adversary has advantage 0. If $|\mathbb{K}| = 1$, then the adversary knows the key, the security is zero, and the adversary succeeds. The adversarial success rate of 1 better reflects the real meaning in this case. The adversarial advantage of 0 does not reflect the adversary's success. Arguably, advantage reflects that when $|\mathbb{K}| = 1$, one has no right to expect security, so advantage is measured against one can expect. That seems to be less useful than our definition of success rate. For slightly larger, but still small $|\mathbb{K}|$, one might expect a similar comparison. To be completed.

4.4 Decisional Active IKA

Definition 4.4.1. *A pair of algorithms* $\mathsf{DAIKA} = (\mathsf{DAIKA}_1, \mathsf{DAIKA}_2)$ *is a decisional active implicit key authentication adversary, if:* DAIKA_1 *takes as input three elements of* $\mathbb{G}$*, and outputs an element of* $\mathbb{G}$ *and an element of some internal state space; and* DAIKA_2 *takes as input an element of some internal state space and an element of the session key space* K*, and outputs a bit. Its success rate* $\sigma(\mathsf{DAIKA})$ *is defined as the expected value of* s *in the following process*

$$A, B, X \leftarrow \mathbb{G} \tag{4.4.1}$$
$$(Y, \Sigma) \leftarrow \mathsf{DAIKA}_1(A, B, X) \tag{4.4.2}$$
$$k \leftarrow \mathbb{K} \vee \mathsf{mqv}(A, B, X, Y) \tag{4.4.3}$$
$$b \leftarrow \mathsf{DAIKA}_2(\Sigma, k) \tag{4.4.4}$$
$$s \leftarrow b \eqcirc (k \triangleq \mathsf{mqv}(A, B, X, Y)) \tag{4.4.5}$$

A decisional active implicit key authentication adversary models an adversary Eve who replaces Bob's ephemeral public key with one of her own choosing. Eve is successful if she is able to learn anything about the Menezes–Qu–Vanstone session key that Alice and Bob compute.

Menezes–Qu–Vanstone key agreement provides decisional active implicit key authentication if no high-success, low-cost adversary DAIKA as in Definition 4.4.1 is constructible.

Remark 4.4.1. A decisional active IKA adversary with success rate $\sigma(\mathsf{DAIKA}) = 0$ is obtained by choosing bit b uniformly at random.

4.5 Relationship Between Grades of IKA

Informally, a passive adversary is a special case of an active adversary that simply opts not to modify Bob's ephemeral public key, and a computational adversary almost immediately gives rise a decisional adversary who simply compares the challenge key with the computed key. Formally, the following four propositions express this.

Proposition 4.5.1. *If* CPIKA *is a computational passive implicit key authentication adversary, then the algorithm* CAIKA *defined by*

$$Y \leftarrow \mathbb{G} \tag{4.5.1}$$
$$\mathsf{CAIKA}(A, B, X) \to (Y, \mathsf{CPIKA}(A, B, X, Y)) \tag{4.5.2}$$

is a computational active implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{CAIKA}) = \sigma(\mathsf{CPIKA})$, *and*
- *computational cost* $\gamma(\mathsf{CAIKA}) \leqslant \gamma(\mathsf{CPIKA}) + (2\log_2(n))(2\gamma(\mathbb{G}) + \gamma(\$))$.

Proof. The main cost CAIKA over CPIKA is the selection $Y \leftarrow \mathbb{G}$, which could be done by selecting $2\log_n(n)$ random bits, to determine a scalar multiplier y, and then computing $Y = yG$ using $2\log_2(n)$ group operations.

Because CAIKA chooses Y uniformly at random, the inputs of CPIKA are four uniformly and independently chosen random group elements. So, the output $\mathsf{CPIKA}(A, B, X, Y)$ equals $\mathsf{mqv}(A, B, X, Y)$ with probability $\sigma(\mathsf{CPIKA})$, and $s = 1$ in the general process (Definition 4.2.1) of CAIKA. □

So CPIKA ⇒ CAIKA. Going from passive to active again can be done in the decisional setting too, so DPIKA ⇒ DAIKA, as in the following formal proposition.

Proposition 4.5.2. *If* DPIKA *is a computational passive implicit key authentication adversary, then the pair of algorithms* $\mathsf{DAIKA} = (\mathsf{DAIKA}_1, \mathsf{DAIKA}_2)$ *defined by*

$$Y \leftarrow \mathbb{G} \tag{4.5.3}$$
$$\mathsf{DAIKA}_1(A, B, X) \to (Y, (A, B, X, Y)) \tag{4.5.4}$$
$$\mathsf{DAIKA}_2((A, B, X, Y), k) \to \mathsf{DPIKA}(A, B, X, Y, k) \tag{4.5.5}$$

is a decisional active implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{DAIKA}) = \sigma(\mathsf{DPIKA})$, *and*
- *computational cost* $\gamma(\mathsf{DAIKA}_1, \mathsf{DAIKA}_2) \leqslant \gamma(\mathsf{DPIKA}) + (2\log_2(n))(\gamma(\mathbb{G}) + \gamma(\$))$.

Proof. The cost analysis is the same as in the proof of Proposition 4.5.1.

The success rate $\sigma(\mathsf{DAIKA})$ is the same as $\sigma(\mathsf{DPIKA})$ because the success indicator formula is the same, and the distribution $Y \leftarrow \mathbb{G}$ ensures that DPIKA has the correct distribution of inputs. □

Computational adversaries can be converted into decisional adversaries. In the passive setting, CPIKA ⇒ DPIKA follows from the next formal proposition.

Proposition 4.5.3. *If* CPIKA *is a computational passive implicit key authentication adversary, then the algorithm* DPIKA *defined by*

$$\mathsf{DPIKA}(A, B, X, Y, k) \to k \stackrel{?}{=} \mathsf{CPIKA}(A, B, X, Y) \tag{4.5.6}$$

is a decisional passive implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{DPIKA}) = \sigma(\mathsf{CPIKA}) - \frac{2}{|\mathbb{K}|}(1 - \sigma(\mathsf{CPIKA}))$, *and*

- *computational cost* $\gamma(\mathsf{DPIKA}) = \gamma(\mathsf{CPIKA})$.

Proof. Algorithm DPIKA invokes CPIKA, which is its main cost, and then does a comparison with k (which will not be counted in the cost), so $\gamma(\mathsf{DPIKA}) = \gamma(\mathsf{CPIKA})$.

The success indicator of the algorithm DPIKA from (4.5.6) above, computed according Definition 4.3.1, will be

$$(k \stackrel{\circ}{=} \mathsf{CPIKA}(A, B, X, Y)) \stackrel{\circ}{-} (k \stackrel{\circ}{=} \mathsf{mqv}(A, B, X, Y)) \tag{4.5.7}$$

Consider the event that $\mathsf{CPIKA}(A, B, X, Y)$ outputs $\mathsf{mqv}(A, B, X, Y)$. This event occurs with probability $\sigma(\mathsf{CPIKA})$ because the inputs A, B, X, Y are independently and uniformly distributed in accordance with Definition 4.1.1. In this case, the success indicator s from (4.5.7) evaluates to 1 because both sides of the operator $\stackrel{\circ}{-}$ are equal.

Otherwise $\mathsf{CPIKA}(A, B, X, Y) \neq \mathsf{mqv}(A, B, X, Y)$ which occurs with probability $1 - \sigma(\mathsf{CPIKA})$ and will be assumed henceforth in this proof.

Consider the event that $k = \mathsf{mqv}(A, B, X, Y)$. This event is independent of the previous event, because k is not an input to CPIKA. This event occurs with probability $\frac{1}{2} + \frac{1}{2|\mathbb{K}|}$. The success indicator (4.5.7) evaluates to $(0 \stackrel{\circ}{-} 1) = -1$ in this case.

Otherwise that $k \neq \mathsf{mqv}(A, B, X, Y)$, which occurs with probability $\frac{1}{2} - \frac{1}{2|\mathbb{K}|}$ and will be assumed henceforth in this proof. In this case k is uniformly distributed among the $|\mathbb{K}| - 1$ value in the set $\mathbb{K} \setminus \{\mathsf{mqv}(A, B, X, Y)\}$.

Since we have assumed that $\mathsf{CPIKA}(A, B, X, Y) \neq \mathsf{mqv}(A, B, X, Y)$, the probability that $k = \mathsf{CPIKA}(A, B, X, Y)$ is $\frac{1}{|\mathbb{K}|-1}$, in which case, s from (4.5.7) evaluates to $(1 \stackrel{\circ}{-} 0) = -1$. Otherwise s evaluates to $(0 \stackrel{\circ}{-} 0) = 1$. Therefore

$$\sigma(\mathsf{DPIKA}) = \sigma(\mathsf{CPIKA}) + (1 - \sigma(\mathsf{CPIKA}))\left(-\left(\frac{1}{2} + \frac{1}{2|\mathbb{K}|}\right) + \left(\frac{1}{2} - \frac{1}{2|\mathbb{K}|}\right)\left(\frac{-1}{|\mathbb{K}| - 1} + \frac{|\mathbb{K}| - 2}{|\mathbb{K}| - 1}\right)\right) \tag{4.5.8}$$

which simplifies to the claimed success rate. □

Remark 4.5.1. If the term in $\sigma(\mathsf{DPIKA})$ with factor $\frac{2}{|\mathbb{K}|}$ is ignored because $|\mathbb{K}|$ will generally be quite large, then the estimate $\sigma(\mathsf{DPIKA}) \approx \sigma(\mathsf{CPIKA})$ follows. This indicates that the definition of success rate appears to have consistent units between the computational and decisional cases. In particular, dividing the decisional success rate by two, so that its total possible range is one, would result in inconsistent units for success rates.

Finally, computational implies decisional in the active setting too, DPIKA ⇒ DAIKA, formalized in the proposition below.

Proposition 4.5.4. *If* CAIKA *is a computational active implicit key authentication adversary, then the pair of algorithms* $\mathsf{DAIKA} = (\mathsf{DAIKA}_1, \mathsf{DAIKA}_2)$ *defined by*

$$(Y, k') \leftarrow \mathsf{CAIKA}(A, B, X) \tag{4.5.9}$$

$$\mathsf{DAIKA}_1(A, B, X) \rightarrow (Y, k') \tag{4.5.10}$$

$$\mathsf{DAIKA}_2(k', k) \rightarrow k' \stackrel{\circ}{=} k \tag{4.5.11}$$

is a decisional active implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{DAIKA}) = \sigma(\mathsf{CAIKA}) - \frac{2}{|\mathbb{K}|}(1 - \sigma(\mathsf{CAIKA}))$, *and*
- *computational cost* $\gamma(\mathsf{DAIKA}_1) + \gamma(\mathsf{DAIKA}_2) \leqslant \gamma(\mathsf{DPIKA})$.

Proof. The total cost of the pair of algorithms is the cost of CAIKA, (and the comparison $k' \stackrel{?}{=} k$ which will be not counted). The success rate is now calculated similarly to the success rate calculation in the proof of Proposition 4.5.3.

The success indicator of the adversary DAIKA above, computed according Definition 4.4.1, will be

$$(k' \stackrel{?}{=} k) \div (k \stackrel{?}{=} \mathsf{mqv}(A, B, X, Y)) \tag{4.5.12}$$

where k' is determined by $(Y, k') = \mathsf{CAIKA}(A, B, X)$. Consider the event that $k' = \mathsf{mqv}(A, B, X, Y)$. This event occurs with probability $\sigma(\mathsf{CAIKA})$ because the inputs A, B, X are independently and uniformly distributed in accordance with Definition 4.2.1. In this case, the success indicator s from (4.5.12) evaluates to 1 because both sides of the operator $\div$ are equal.

Otherwise $k' \neq \mathsf{mqv}(A, B, X, Y)$ which occurs with probability $1 - \sigma(\mathsf{CAIKA})$ and will be assumed henceforth in this proof.

Consider the event that $k = \mathsf{mqv}(A, B, X, Y)$. This event is independent of the previous event, because k is not an input to CAIKA. This event occurs with probability $\frac{1}{2} + \frac{1}{2|\mathbb{K}|}$. The success indicator (4.5.12) evaluates to $(0 \div 1) = -1$ in this case.

Otherwise $k \neq \mathsf{mqv}(A, B, X, Y)$, which occurs with probability $\frac{1}{2} - \frac{1}{2|\mathbb{K}|}$ and will be assumed henceforth in this proof. In this case, k will be uniformly distributed among the $|\mathbb{K}| - 1$ elements in the set $\mathbb{K} \setminus \{\mathsf{mqv}(A, B, X, Y)\}$.

Because we have assumed that $k' \neq \mathsf{mqv}(A, B, X, Y)$, we will have $k' \in \mathbb{K} \setminus \{\mathsf{mqv}(A, B, X, Y)\}$, and so, with probability $\frac{1}{|\mathbb{K}|-1}$, it will hold that $k = k'$, in which case s from (4.5.12) evaluates to $(1 \div 0) = -1$. Otherwise, s evaluates to $(0 \div 0) = 1$. Therefore

$$\sigma(\mathsf{DAIKA}) = \sigma(\mathsf{CAIKA}) + (1 - \sigma(\mathsf{CAIKA}))\left(-\left(\frac{1}{2} + \frac{1}{2|\mathbb{K}|}\right) + \left(\frac{1}{2} - \frac{1}{2|\mathbb{K}|}\right)\left(\frac{-1}{|\mathbb{K}| - 1} + \frac{|\mathbb{K}| - 2}{|\mathbb{K}| - 1}\right)\right) \tag{4.5.13}$$

which simplifies to the claimed success rate. □

The overall relationship between the various grades is thus:

$$\begin{array}{ccc} \mathsf{CAIKA} & \Rightarrow & \mathsf{DAIKA} \\ \Uparrow & & \Uparrow \\ \mathsf{CPIKA} & \Rightarrow & \mathsf{DPIKA} \end{array} \tag{4.5.14}$$

with the double arrows again indicating the ability to convert an adversary at the tail of the arrow into the adversary at the head of the arrow. Resistance to adversaries, therefore of course, flows in the direction opposite to the arrows.

More generally, relations to the logical composite adversaries CAIKA&DPIKA and CAIKA|CPIKA, as follows:

$$\begin{array}{ccccccc} & & \mathsf{CAIKA} & \Rightarrow & \mathsf{CAIKA|DPIKA} & \Rightarrow & \mathsf{DAIKA} \\ & & \Uparrow & & \Uparrow & & \\ \mathsf{CPIKA} & \Rightarrow & \mathsf{CAIKA\&DPIKA} & \Rightarrow & \mathsf{DPIKA} & & \end{array} \tag{4.5.15}$$

If (CAIKA&DPIKA) ⇒ CPIKA, or DAIKA ⇒ (CAIKA|DPIKA), then (4.5.15) would look more like (4.5.14).

4.6 Significance of the Four Grades of IKA Adversary

Four formal grades of implicit key authentication adversary have been defined in this report. Mathematically, definitions are neither right nor wrong; cryptologically, definitions attempt to model some real world threats. So, it is worthwhile to compare the significance of considering each of the four grade.

4.6.1 Scope Over Prudence

A strong tradition in cryptology is prudence. When defining (or just contemplating) an adversary, prudence has two aspects:

1. The adversary should be considered to have won after having gaining the slightest benefit, such as a single bit of information that is intended to be secret.

2. The adversary should be considered to have extensive capabilities to intercept, interfere, and perhaps compute.

Prudence would attach greater significance to the decisional active grade adversary implicit key authentication adversary. Decisional is more prudent than computational because of the first aspect above. Active is more prudent that passive because of the second aspect above.

Formally, security against the prudent choice of adversary implies security against the less prudent choice. So, therefore, not only would the most prudent course appear to be to concentrate on decisional active implicit key authentication, but furthermore one can safely ignore the other three grades, because the security against the most prudent grade ensures security against the less prudent grades.

Yet a stronger tradition in cryptology is scope. The adversaries to cryptographic schemes are generally limited by some pre-arranged scope of the scheme, usually reflecting the intent of the scheme. For example, a digital signature is not intended to keep the signed message hidden. Similarly, a ciphertext is not intended to provided non-repudiation.

Scope is a stronger condition than prudence because although more prudent security definitions exist, such as the definition for signcryption, but are not routinely applied when deemed out of scope.

A strategy of merging prudence and scope leads to circular reasoning. For example, given that this report is analyzing Menezes–Qu–Vanstone, it could argue that scope excludes anything that Menezes–Qu–Vanstone cannot accomplish, such as non-repudiation. If one also chooses the most prudent security definition within this scope, then assuming that one could correctly identify the most prudent security definition, then Menezes–Qu–Vanstone would certainly meet its security definition. Such circular reasoning could be applied to any other cryptographic scheme, so every scheme would be secure.

This circular reasoning can be obscured by defining a class of cryptographic schemes. For example, in the case of Menezes–Qu–Vanstone, this class could be all two-pass key agreement schemes. If one defines the security definition for this class of schemes based on the most prudent possible definition for Menezes–Qu–Vanstone, then one obtains artificial comparison between Menezes–Qu–Vanstone and other schemes. Similarly, if one defines security based on another scheme in this class, one obtain yet another artificial comparison.

So, security definitions cannot be evaluated solely on the basis of scope and prudence. Intent must be used to set the scope, and realism of threats must be used to set the level of prudence. Generally, users of a cryptographic scheme must be aware of the intended security features. Users who adhere to these intended purposes of the scheme. Any realistic potential harms to the user due to the failure of the scheme to meet its intended purposes should be deemed a security flaw.

The following subsections try to sort out what the most important grade of implicit key authentication under the intended purpose of two-pass Menezes–Qu–Vanstone key agreement, under the rationale of realistic harm. (Perhaps similar rationale could be applied to assess more advanced security features such as forward secrecy.)

4.6.2 Computations Over Decisions

Suppose that a computational (either passive or active) implicit key authentication adversary to Menezes–Qu–Vanstone could be constructed. This adversary could determine the Menezes–Qu–Vanstone session key. Whatever symmetric key encryption algorithm the Menezes–Qu–Vanstone session key is used for, the adversary could compromise its use. If the session key is used as an AES key to encrypt some message, then a computational implicit key authentication adversary to Menezes–Qu–Vanstone could decrypt the message, without having to attack AES.

Suppose that a decisional, either active or passive, implicit key authentication adversary to Menezes–Qu–Vanstone could be constructed. Essentially, such an adversary could learn up to one bit of the Menezes–Qu–Vanstone session key. The impact of this leaked bit depends on the which the following two uses of session key are made.

1. The Menezes–Qu–Vanstone session key could be used as the secret key for symmetric key algorithms, which is the intended use of MQV session keys. Most symmetric key algorithms are fairly robust. If the adversary knows one bit of the symmetric key, this does not substantially help the adversary attack the symmetric key algorithm. For example, if the MQV session key was used as an AES encryption key, and an adversary learned one bit of the MQV session key, could the adversary learn anything about the encrypted message? If so, this would seem to be new weakness discovered in AES: a one bit leak in the AES key would result in a one bit leak in the AES ciphertext. Information-theoretically, this is possible, but still it seems to be a computationally hard problem.

2. The Menezes–Qu–Vanstone session key could be used as the key stream for one-time pad, which is not the intended use of the MQV session key. In this use case, a single bit, or less, of the session key known to the adversary could reveal the value of the message to the adversary. (More precisely, it could reveal whether a ciphertext encrypts one of two messages, both known to the adversary.) Therefore, provided that MQV session key are not used as one-time pads, computational adversary are overwhelmingly more important than decisional adversaries.

Remark 4.6.1. Decisional implicit key authentication may be valuable in going part of way towards establishing some kind of composability, in the sense that the MQV session key can be used even in protocols that might be weak. But more typically, composability would presume that the protocol using the MQV session key is strong, not weak. Rather, universal composability would try to establish that the composition of two strong protocols is not somehow weak, which really seems to be an orthogonal property to decisional implicit key authentication.

Computational implicit key authentication adversaries are always more important for Menezes–Qu–Vanstone than the decisional adversaries, which, for the intended uses of Menezes–Qu–Vanstone session keys appear to have no practical consequences.

If Menezes–Qu–Vanstone session keys are used, contrary to intended usage, as one-time pads, or if symmetric key algorithms that turn to have unforeseen security weaknesses, then decisional implicit key authentication adversaries can have practical consequences. So, decisional adversaries should not be ignored entirely, because of these two possibilities.

Alternatively, some encryption algorithms are essentially stream ciphers, generating a keystream from a key, with the keystream used as one-time pad combined with the plaintext. If we model that keystream generation step as part of the key derivation function, then the decisional variants of implicit key authentication become much more crucial.

4.6.3 Active Over Passive

Interception and replacement of electronic communication is possible but not free. Therefore, an active adversary has extra cost over a passive adversary. In Menezes–Qu–Vanstone key agreement, the ephemeral keys are not authenticated, so active adversaries must be considered.

Indeed, if only passive adversaries are considered, then this would be essentially assuming that the ephemeral keys are authenticated. From an implicit key authentication perspective, then one might as well merge the static and ephemeral keys. Consequently Menezes–Qu–Vanstone would amount to nothing much more than authenticated (static) Diffie–Hellman key agreement.

The possibility of active attack opens up the possibility that the implicit key authentication of Menezes–Qu–Vanstone could be worse than the implicit key authentication of authenticated Diffie–Hellman key agreement. Such static Diffie–Hellman has no unauthenticated ephemeral keys which could be tampered with, so is not at all subject to active implicit key authentication attacks. Menezes–Qu–Vanstone uses unauthenticated ephemeral keys, so it is open to possible active attacks.

This raises the natural question: why does Menezes–Qu–Vanstone use unauthenticated ephemeral keys if it opens a door to possible active attacks? The answer is that ephemeral keys make possible other security goals, such as forward secrecy. Forward secrecy is not examined in this report. Ephemeral public keys are generally not pre-authenticated (e.g. certified) because they are used too-often used and too short-lived. So, as a practicality, ephemeral key, which are necessary for forward secrecy, must be necessarily remain unauthenticated. Or rather, ephemeral keys become authenticated in real time, usually by application of pre-authenticated static keys and the Menezes–Qu–Vanstone key agreement scheme.[18]

So potential security gains, such as forward secrecy, of using (unauthenticated) ephemeral public keys, must be weighed against the potential security losses due to the ephemeral key being unauthenticated. By considering active implicit key authentication attacks, this report examines the potential consequences upon implicit key authentication, the primary goal of key agreement, of Menezes–Qu–Vanstone using unauthenticated ephemeral keys. In other words, this reports tries to answer whether the baby has been thrown out with the bath water: has IKA been compromised in MQV to achieve forward secrecy and efficiency?

4.6.4 Security Spectrum and Conjecture Minimization

Cryptology commonly counts on conjectures. In traditional cryptology, certain conjectures (such that it is hard to solve a discrete logarithm) must hold or else a cryptographic scheme is broken. In provable security, if certain conjectures hold then it can be concluded that a cryptographic scheme is secure. Ideally, the conjectures that cryptology relies on should be minimal.

Having a spectrum of security properties, such as the four grades of implicit key authentication, makes it possible to individually minimize the conjectures for each security property in the spectrum.

If instead one focuses on the most prudent security property in the spectrum, say decisional active implicit key authentication, the conjectures on which this properties rely might be rather strong conjectures, and thereby provide one with rather weak assurance. One might not realize that the more crucial security properties, such as computational passive implicit key authentication, rely on much weaker conjectures, and thereby provide one with much stronger assurance.

[18] This is not to suggest that the ephemeral keys should be subsequently used as though they were static keys. Indeed, this should normally be impossible, because the ephemeral private keys are destroyed after use.

5 Conditions on the Components of Menezes–Qu–Vanstone

In this section, various conditions on the three components of Menezes–Qu–Vanstone key agreement are formally defined and related.

Remark 5.0.2. The reason to consider these conditions are primarily due to the relationship of these conditions to the capacity of Menezes–Qu–Vanstone to provide implicit key authentication. These relations will be examined in §6 and §7.

Remark 5.0.3. Many of the conditions, such as the Diffie–Hellman problem being hard, are already well-known and well-studied conditions. Similarly, if key derivation functions are viewed as hash functions, then many of the conditions and their relations have been studied before, such as by Rogaway and Shrimpton [RS04] (who mainly addresses keyed hash functions).

Remark 5.0.4. Skipping this section, returning only to it as needed via cross-references, should, it is hoped, not reduce the intelligibility of the report.

5.1 Classes of Conditions

Some conditions apply to a single component, and we will call them *isolated* conditions, at least in general contexts about conditions. Isolated conditions on the group are considered in §5.2; isolated conditions on the conversion function are considered in §5.3; and isolated conditions on the key derivation function are considered in §5.4.

Remark 5.1.1. The conversion function and the key derivation function are both function defined with domain $\mathbb{G}$. A condition regarding these functions will be considered not to structurally involve the component $\mathbb{G}$ if the condition is statable over domains other than $\mathbb{G}$.

A condition may apply to two components. Such a condition will be called a *joint* condition. Joint conditions on the group and conversion function are considered in §6.5. Joint conditions on the group and key derivation function are considered §6.6.

Given two conditions, another condition is their logical disjunction, which is the requirement at least one of two conditions hold. Similarly, the logical conjunction is the condition that both of the two given conditions hold. Logical conjunctions and logical disjunction of conditions will be called compound conditions. The given conditions used to form the compound condition are its constituent conditions.

A joint conditions may be a compound condition of two constituent isolated conditions. Such a joint condition will be called a *separable* condition. The constituent isolated conditions will be called the component conditions of the joint condition. Because separable conditions are expressible as logical combinations of their component conditions, they may not be given separate formal definitions.

A condition may apply to all three components, and such a condition will be called a *complex* condition. The four grades of implicit key authentication are each complex conditions. These have already been defined in §4. Other complex conditions are considered in §5.7.

As with joint conditions, a complex condition may separate into a logical combination of three isolated conditions, or into a logical combination of a joint condition and an isolated condition. As with joint conditions, separable complex conditions may not be given separate definitions.

This section will relate various conditions, excluding the four conditions corresponding to the four grades of implicit key authentication, which will be related to other conditions in §6 and §7.

5.2 Isolated Conditions on the Group

Three isolated conditions on the group component of Menezes–Qu–Vanstone are defined and related in this section. These conditions, and the relations between them, are well-known.

Remark 5.2.1. The three conditions are that three problems are hard: the discrete logarithm problem, the Diffie–Hellman problem, and the decisional Diffie–Hellman problem.

Remark 5.2.2. Other group-isolated conditions may be related the implicit key authentication of MQV. Further research may relate such conditions to IKA of MQV.

Remark 5.2.3. The particular two group–isolate problems: the Gap Diffie–Hellman problem, and the one-more discrete logarithm problem are familiar to me, but a connection to the IKA of MQV did not occur to me.

These two problems are oracle-assisted problems. This report considers other oracle-assisted problems and their relations to the IKA of MQV, but the oracles involved relate to the derivation function, so the resulting problems are not group-isolated.

5.2.1 Discrete Logarithms Hard

Definition 5.2.1. *An algorithm* DL *is a discrete logarithm solver in* $\mathbb{G}$ *to the base* G*, if it takes as input an element of* $\mathbb{G}$ *and outputs an integer in* $\mathbb{Z}_n$*. Its success* $\sigma(\mathsf{DL})$ *rate is defined as the expected value of* s *in the following process*

$$Q \leftarrow \mathbb{G} \tag{5.2.1}$$
$$q \leftarrow \mathsf{DL}(Q) \tag{5.2.2}$$
$$s \leftarrow Q \stackrel{?}{=} qG \tag{5.2.3}$$

The hardness condition associated with the discrete logarithm solver is that a high-success, low-cost discrete logarithm is not constructible.

Remark 5.2.4. Lemma 6.2.1 shows that this condition is necessary for Menezes–Qu–Vanstone to provide computational passive implicit key authentication.

Remark 5.2.5. Nechaev [Nec94] introduced the generic group model, which Shoup [Sho97] extended. In this model, the group $\mathbb{G}$ is provided by an oracle. Nechaev and Shoup proved that when DL is restricted to the generic group model, meaning that it must access the group $\mathbb{G}$ via an oracle, algorithm DL requires about $\sqrt{n}$ group operations to have a reasonable success rate.

Remark 5.2.6. The fastest known discrete logarithm solvers DL that are defined over any $\mathbb{G}$, are Shanks' baby-step-giant-step and Pollard's rho and lambda methods, are generic group algorithms. Baby-step-giant-step is a generic group algorithm in the sense of Nechaev's model. Pollard's rho algorithm, and lambda algorithm, are generic group algorithm in the sense of Shoup's model.

Each of these take approximately $\sqrt{n}$ group operations to obtain a reasonable success rate, which is close to optimal by the lower bounds of Nechaev and Shoup.

Remark 5.2.7. For certain groups $\mathbb{G}$, specific discrete logarithm solvers DL are known that are faster than the generic algorithms. For example:

- When $\mathbb{G} = \mathbb{Z}_n$, the integers modulo n, the Euclidean algorithm allows us to construct $H \equiv G^{-1} \bmod n$, and then $\mathsf{DL}(Q) \to QH$ is a discrete logarithm solver with success rate 1. Its existential computational cost is one multiplication modulo n. Its constructive computational cost is one multiplication modulo n and one inversion modulo n.

- When $\mathbb{G}$ is the group of integers modulo n under multiplication (not addition as above), then algorithms known as index-calculus algorithms, such as the number field sieve, are faster than generic algorithms.
- When $\mathbb{G}$ is an elliptic curve group defined over a finite field $\mathbb{F}_q$, certain exceptional elliptic curve groups have discrete logarithm solvers significantly faster than the generic discrete logarithm solvers. Some examples of exceptional groups are given below.
  - Elliptic curve groups that have $n = q$ succumb to the Semaev–Smart–Satoh–Araki attack.
  - Elliptic curve groups that have $n|q^b - 1$ for some small rational b whose denominator divides the degree of $\mathbb{F}_q$, succumb to the Menezes–Okamoto–Vanstone and Frey–Rück attacks.
  - Certain elliptic curve groups over certain extension finite fields where $q = p^m$ for some composite degree m succumb to a class of attacks known as Weil descent attacks.

  Otherwise, the fastest known discrete logarithm are only slightly faster than the generic algorithms, with the improvement arising from some group operations, such as negation or certain endomorphism, being more efficient than general group operations.

Remark 5.2.8. Most standards for elliptic curve Menezes–Qu–Vanstone key agreement exclude the known exceptional elliptic curve groups above whose discrete logarithm problem can be solved faster than with generic group algorithm such as Pollard's rho algorithm.

The following conjecture, which would ensure the hardness condition for DL, has often been assumed.

Conjecture 5.2.2. *Any discrete logarithm solver* DL *in an non-exceptional elliptic curve curve group is such that*

$$\sigma(\mathsf{DL}) \leqslant \frac{\theta}{n}\binom{\gamma(\mathsf{DL})/\gamma(\mathbb{G})}{2} \tag{5.2.4}$$

where $\theta \leqslant 1$ *is a value close to* 1.

Remark 5.2.9. The conjecture above claims the discrete logarithm is existentially hard. This may be harder than is actually needed for security. If the discrete logarithm problem is constructibly hard, but existentially easy, then security does not seem to be compromised.

Remark 5.2.10. Equation 5.2.4 may be reformulated approximately as

$$\frac{\sigma(\mathsf{DL})}{\gamma(\mathsf{DL})^2} \leqslant \frac{1}{2n\gamma(\mathbb{G})^2}. \tag{5.2.5}$$

Unlike Remark 4.1.3, it is not the ratio σ/γ that is bounded, but rather σ/γ^2 that is bounded.

Remark 5.2.11. The discrete logarithm problem is a computational problem. It has a decisional variant, but the decisional variant is generally an easy problem, because the exponentiation can be used to make the decision.

5.2.2 Diffie–Hellman Problem Hard

Definition 5.2.3. *An algorithm* DH *is a Diffie–Hellman problem solver in* $\mathbb{G}$ *to the base* G*, if it takes as input two elements of* $\mathbb{G}$ *and outputs one element of* $\mathbb{G}$*. Its success rate* $\sigma(\mathsf{DH})$ *is defined as the expected value of* s *in the following process*

$$P, Q \leftarrow \mathbb{G} \tag{5.2.6}$$
$$R \leftarrow \mathsf{DH}(Q) \tag{5.2.7}$$
$$s \leftarrow R \stackrel{?}{=} P \otimes Q \tag{5.2.8}$$

The hardness condition associated with the Diffie–Hellman problem is that a high-success, low-cost Diffie–Hellman problem solver is not constructible.

Remark 5.2.12. Lemma 6.2.2 shows that this condition is necessary for Menezes–Qu–Vanstone to provide computational passive implicit key authentication.

A well-known fact is that, for the Diffie–Hellman problem to be hard, the discrete logarithm problem must be hard, which the following proposition re-states.

Proposition 5.2.4. *If* DL *is a discrete logarithm solver, then the algorithm* DH *defined by*

$$\mathsf{DH}(P,Q) \to \mathsf{DL}(Q)P \tag{5.2.9}$$

is a Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{DH}) = \sigma(\mathsf{DL}) + \frac{1}{n}(1-\sigma(\mathsf{DL}))$, *and*
- *computational cost* $\gamma(\mathsf{DH}) \leqslant \gamma(\mathsf{DL}) + (2\log_2(n))\gamma(\mathbb{G})$.

Proof. Algorithm DH invokes DL and also does a scalar multiplication in the group $\mathbb{G}$, so the cost is bounded as claimed.

By definition of the general process for DH (Definition 5.2.3), the element Q is uniformly distributed in $\mathbb{G}$. Let q be the output of $\mathsf{DL}(Q)$. Let R be the output of DH algorithm (5.2.9), so that $R = qP$.

The success indicator for DH is $R \mathrel{\mathring{=}} P \otimes Q$. Since $P \otimes G = P$, it follows $R = qP = P \otimes qG$. The success indicator of DH is thus $P \otimes qG \mathrel{\mathring{=}} P \otimes Q$. If π is function $\pi(U) = P \otimes U$ the success indicator of DH is obtained by applying π to both sides of the operator $\mathrel{\mathring{=}}$ in the success indicator of DL (which is $qG \mathrel{\mathring{=}} Q$).

If $P \neq 0$, then π is bijective, and its application to both sides of the operator $\mathrel{\mathring{=}}$ does not change the outcome of the operator. In other words, in this case, the success indicator for DH equals that of DL.

In the case $P = 0$, algorithm DH above always has success indicator 1 (even if DL has success indicator 0). Therefore $\sigma(\mathsf{DH}) = \frac{1}{n} + \sigma(\mathsf{DL})(1-\frac{1}{n})$, which equals the claimed success rate. □

Remark 5.2.13. Over general groups $\mathbb{G}$, the fastest known way in general to solve the Diffie–Hellman problem is to solve a discrete logarithm, much like that algorithm in in Proposition 5.2.4.

Remark 5.2.14. In Nechaev and Shoup's generic group model, solving the Diffie–Hellman problem with sufficiently high success rate also has computational cost of at least about $\sqrt{n}$ group operations.

Remark 5.2.15. Maurer, Wolf, Boneh and Lipton made significant progress on the converse of the relation above between DLP and DHP. Informally, for the DLP to be hard, the DHP must be hard (in general). Their results are not obvious, and involve the existence of auxiliary elliptic curve defined of the field $\mathbb{Z}_n$. The result depends on the order of this elliptic curve having no large prime factor. Muzereau, Smart and Vercauteren constructed Maurer–Wolf auxiliary curves for most of the 15 elliptic curve groups recommended by NIST in FIPS 186-3.

Remark 5.2.16. The proof of Proposition 5.2.4 does not use the fact that P is uniformly distributed in Definition 5.2.5. So, if DH' is a modification of DH that is only required to work for P with non-uniform distribution (perhaps even a fixed P, which is an existentially easy problem to solve, but appears to be constructibly difficult). The proof of Proposition 5.2.4 implies that $\mathsf{DL} \Rightarrow \mathsf{DH}'$. A more indirect proof would result if $\mathsf{DH} \Rightarrow \mathsf{DH}'$ because, then, $\mathsf{DL} \Rightarrow \mathsf{DH} \Rightarrow \mathsf{DH}'$, and of the transitivity of $\Rightarrow$. The reduction $\mathsf{DH} \Rightarrow \mathsf{DH}'$ may be obtained by turning the problem presented to DH' into the one presented to DH by blinding, which is scalar multiplication of P by a random integer, say β, and later unblinding the result of DH to obtain the result of DH.

There are results in the opposite direction. Algorithm DH' may applied to the problem of DH, but its success rate would need correcting dependent upon on how well a uniform distribution of P supplied to DH matches the

required distribution for DH', say by measure μ. Repeating $1/\mu$ runs DH' could yield a DH with similar success rate to DH'. This reduction may be an instance of what is know as random self-reducibility. This may be regarded as a weak form of $\mathsf{DH}' \Rightarrow \mathsf{DH}$. Maurer, Wolf, Boneh and Lipton's results may be regarded as a weak form of $\mathsf{DH} \Rightarrow \mathsf{DL}$. The composition of these results give a weak form $\mathsf{DH}' \Rightarrow \mathsf{DL}$, depending on the factor μ.

A more direct, and stronger, result of the form $\mathsf{DH}' \Rightarrow \mathsf{DL}$ is provided by the results of Brown and Gallant, and of Cheon. These results depend on the factorization of $n-1$ and $n+1$, respectively, but unlike the previous results, work for very low μ, such as when P is fixed.

The problem DH' does not seem immediately relevant to the security of Menezes–Qu–Vanstone key agreement because the ephemeral keys are randomized, and all the effective results using DH' involve repeated uses of the same private key.

5.2.3 Decisional Diffie–Hellman Problem Hard

Definition 5.2.5. *An algorithm* DDH *is a decisional Diffie–Hellman problem solver in* $\mathbb{G}$ *to the base* G*, if it takes as input three elements of* $\mathbb{G}$ *and outputs a bit. Its success rate* $\sigma(\mathsf{DDH})$ *is defined as the expected value of* s *in the following process*

$$P, Q \leftarrow \mathbb{G} \tag{5.2.10}$$
$$R \leftarrow \mathbb{G} \vee (P \otimes Q) \tag{5.2.11}$$
$$b \leftarrow \mathsf{DDH}(P, Q, R) \tag{5.2.12}$$
$$s \leftarrow b \eqcirc (R \stackrel{\circ}{=} P \otimes Q) \tag{5.2.13}$$

The hardness condition associated with decisional Diffie–Hellman solver is that a high-success, low-cost DDH is not constructible.

Remark 5.2.17. This report has not identified this isolated condition as necessary on its own for any form implicit key authentication of Menezes–Qu–Vanstone key agreement.

Conjecture 5.5.15 and Lemma 6.6.2 together suggest that a hard decisional Diffie–Hellman problem partially-necessary (§6.1) for decisional passive implicit key authentication.

It is well-known that the computational Diffie–Hellman problem must be hard if it is to be true that the decisional Diffie–Hellman problem is hard. The following proposition merely re-states so.

Proposition 5.2.6. *If* DH *is a Diffie–Hellman problem solver, then the algorithm* DDH *defined by*

$$\mathsf{DDH}(P, Q, R) \to R \stackrel{\circ}{=} \mathsf{DH}(P, Q) \tag{5.2.14}$$

is a decisional Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{DDH}) = \sigma(\mathsf{DH}) - \frac{1}{n}(1 - \sigma(\mathsf{DH}))$*, and*
- *computational cost* $\gamma(\mathsf{DDH}) = \gamma(\mathsf{DH})$.

Proof. The computational cost $\gamma(\mathsf{DDH})$ is as claimed because algorithm DDH just invokes DH and does a comparison (which will not be counted in the cost).

In the process (Definition 5.2.5) for our algorithm DDH from (5.2.14) above, the success indicator will be

$$(R \stackrel{\circ}{=} \mathsf{DH}(P, Q)) \eqcirc (R \stackrel{\circ}{=} P \otimes Q) \tag{5.2.15}$$

In the process (Definition 5.2.3) for DH, consider the event that $\mathsf{DH}(P, Q) = P \otimes Q$, which happens with probability $\sigma(\mathsf{DH})$ because the inputs P and Q that DDH supplies to DH are independent and uniformly

distributed in accordance with process for DH. This causes (5.2.15) to evaluate to 1, because both sides of the operator $\div$ are equal.

Otherwise, the complementary event $\mathsf{DH}(P,Q) \neq P \otimes Q$ occurs, which happens with probability $1 - \sigma(\mathsf{DH})$, which will now be assumed.

Consider the event $R = P \otimes Q$, which happens with probability $\frac{1}{2} + \frac{1}{2n}$. This event is independent of the previous event, because the choice of R is determined before DH is invoked, and the R is not given to DH as an input. In this event, (5.2.15) evaluates to $(0 \div 1) = -1$.

Otherwise the $R \neq P \otimes Q$, occurs, which has probability $\frac{1}{2} - \frac{1}{2n}$. In this event, (5.2.15) evaluates to $(0 \div 0) = 1$. Therefore,

$$\begin{aligned} \sigma(\mathsf{DDH}) &= \sigma(\mathsf{DH}) + (1 - \sigma(\mathsf{DH}))\left(-\left(\frac{1}{2} + \frac{1}{2n}\right) + \left(\frac{1}{2} - \frac{1}{2n}\right)\right) \\ &= \sigma(\mathsf{DH}) - \tfrac{1}{n}(1 - \sigma(\mathsf{DH})) \end{aligned} \tag{5.2.16}$$

as desired. □

Remark 5.2.18. The converse of the relation above between DHP and DDHP is known to fail in certain groups, especially pairing-friendly elliptic curve groups. In these groups, the DDHP is easy to solve, but the best known algorithms to the solve the DHP take approximately $\sqrt{n}$ group operations.

5.3 Isolated Conditions on the Conversion Function

5.3.1 Conversion Function Predicting Hard

Definition 5.3.1. *An algorithm* CFP *is a conversion function predictor for conversion function* $f : \mathbb{G} \to \mathbb{Z}_n$ *if it takes no inputs and outputs an integer. Its success rate* $\sigma(\mathsf{CFP})$ *is the expected value of* s *in the following process*

$$j \leftarrow \mathsf{CFP}() \tag{5.3.1}$$
$$R \leftarrow \mathbb{G} \tag{5.3.2}$$
$$s \leftarrow j \stackrel{\circ}{=} f(R) \tag{5.3.3}$$

The hardness condition associated with a conversion function predictor is that a high-success low-cost conversion function predictor is not constructible.

Remark 5.3.1. Lemma 6.3.1 shows that this condition is necessary for Menezes–Qu–Vanstone to provide computational active implicit key authentication.

Remark 5.3.2. A general algorithm for CFP is as follows:

$$P \leftarrow \mathbb{G} \tag{5.3.4}$$
$$\mathsf{CFP}() \to f(P) \tag{5.3.5}$$

If there exists any conversion function predictor with success rate σ, then the specific CFP has success rate at least σ^2.

In this instance, a relation between extistential and constructible hardness of problems can be stated.

Remark 5.3.3. Another general algorithm, extending that in Remark 5.3.2, is

$$P_1, \ldots, P_m \leftarrow \mathbb{G} \tag{5.3.6}$$
$$\mathsf{CFP} \to \mathsf{mode}(f(P_1), \ldots, f(P_m)) \tag{5.3.7}$$

where mode is an algorithm such that $\mathsf{mode}(j_1, \ldots, j_m)$ outputs a value j_i with the property that $|\{j_h : j_h = j_i\}| \geqslant \{j_h : j_h = j_g\}$ for all $g \in \{1, 2, \ldots, m\}$.

It may be an interesting exercise to find a lower bound on the success rate of this CFP compared to optimal success rate over all possible conversion function predictors. When f has one large pre-image and many small pre-images, then this algorithm appears may have a significantly higher success than the algorithm from Remark 5.3.2.

Standardized elliptic curve MQV conversion functions f are specified with the property that, for each j, the probability that $f(R) = j$, over random R, seems to be approximately $1/\sqrt{n}$. Therefore,

Conjecture 5.3.2. *If f is the standardized elliptic curve MQV conversion function and* CFP *conversion function predictor for f, then its success rate $\sigma(\mathsf{CFP})$ is at most approximately $1/\sqrt{n}$.*

Remark 5.3.4. In this case of the standardized elliptic curve MQV conversion function, algorithm CFP from Remark 5.3.2 heuristically has success rate about $1/\sqrt{n}$, which is higher than the lower bound provided in Remark 5.3.2 because the pre-image sizes of this conversion function are quite uniform.

Remark 5.3.5. In this case of the standardized elliptic curve MQV conversion function, algorithm CFP from Remark 5.3.3 heuristically has success rate about $1/\sqrt{n}$, which is not much higher than the success rate of the algorithm from Remark 5.3.2 because the pre-image sizes of this conversion function are quite uniform.

5.4 Isolated Conditions on the Key Derivation Function

Key derivation functions are generally constructed from general purpose hash functions.

Remark 5.4.1. This report has tried to identify security conditions for key derivation functions that are relevant to the capacity of Menezes–Qu–Vanstone to provide implicit key authentication.

It so happened, whether due to some underlying fundamental reason or just due to lack of imagination, that the conditions identified in this report appear to almost exactly coincide with traditional conditions for hash functions.

So, it should be possible to take some known security definitions for hash functions, such the Rogaway–Shrimpton definitions [RS04], and find a correspondence between those definitions with the definitions here.

Furthermore, it should be possible to take existing hash functions, such as SHA-256, that are believed to be secure hash function under the existing security definitions for hash function, and build key derivation from them in a way that preserves the security definitions useful for Menezes–Qu–Vanstone key agreement. Indeed, perhaps the standardized key derivation functions constructions already preserve the requisite security conditions.

5.4.1 Key Derivation Function Predicting Hard

Definition 5.4.1. *An algorithm* KDP *is a key derivation predictor for key derivation function $d : \mathbb{G} \to K$ if it takes no inputs and outputs an element of $\mathbb{K}$ (a possible session key). Its success rate $\sigma(\mathsf{KDP})$ is the expected value of s in the following process*

$$k \leftarrow \mathsf{KDP}() \tag{5.4.1}$$
$$U \leftarrow \mathbb{G} \tag{5.4.2}$$
$$s \leftarrow k \stackrel{\circ}{=} d(U) \tag{5.4.3}$$

The hardness condition associated with a key derivation function predictor is that a high-success low-cost key derivation function predictor is not constructible.

Remark 5.4.2. Key derivation function predictor-resistance is necessary if Menezes–Qu–Vanstone is to provide computational passive implicit key authentication because of Lemma 6.4.1.

Remark 5.4.3. A general algorithm for a key derivation function predictor is the algorithm

$$P_1, \ldots, P_m \leftarrow \mathbb{G} \tag{5.4.4}$$
$$\mathsf{KDP} \rightarrow \mathsf{mode}(d(P_1), \ldots, d(P_m)) \tag{5.4.5}$$

This algorithm KDP is just the algorithm CFP from Remark 5.3.3, with f replaced by d.

Remark 5.4.4. It would be interesting to estimate the success rate of the algorithm KDP from Remark 5.4.3 under the heuristic assumption that $d : \mathbb{G} \to \mathbb{K}$ is a random function. A reasonable guess may be that, when $|\mathbb{K}| \gg n$, the success rate would be roughly $1/n$, but, when $|\mathbb{K}| \ll n$, the success rate would be roughly $1/|\mathbb{K}|$.

Remark 5.4.5. An issue with the key derivation predictor is that its success indicator is not determined solely by its inputs and outputs, but rather by an experiment variable about which no information is supplied as input to the predictor.

Algorithms that use a key derivation predictor may rely on this experiment variable, but only with implicit knowledge, and therefore may not be able test the success of a particular application the predictor.

5.4.2 Key Derivation Function Distinguishing Hard

Definition 5.4.2. *An algorithm* KDD *is a key derivation distinguisher for key derivation function* $d : \mathbb{G} \to \mathbb{K}$ *if it takes an input from the session key space* $\mathbb{K}$ *and outputs a bit. Its success rate* $\sigma(\mathsf{KDD})$ *is the expected value of* s *in the following process*

$$U \leftarrow \mathbb{G} \tag{5.4.6}$$
$$k \leftarrow \mathbb{K} \vee d(U) \tag{5.4.7}$$
$$b \leftarrow \mathsf{KDD}(k) \tag{5.4.8}$$
$$s \leftarrow b \eqcirc (k \stackrel{\circ}{=} d(U)) \tag{5.4.9}$$

A hardness condition associated with the key derivation function distinguisher is that a high-success low-cost key derivation distinguisher above is not constructible. When this condition holds, the key derivation function is said to be indistinguishable.

Remark 5.4.6. Indistinguishability of the key derivation function is a necessary condition for Menezes–Qu–Vanstone to provide decisional passive implicit key authentication because of Lemma 6.4.2.

Remark 5.4.7. If $\mathbb{K} = \mathbb{G}$ and d is bijective, then the key derivation function d is indistinguishable. In most cases, $\mathbb{G} \neq \mathbb{K}$ because of the various other requirements for the group $\mathbb{G}$ and key space $\mathbb{K}$. Also, even if $\mathbb{K} = \mathbb{G}$, the identity function may not be ideal for security.

Remark 5.4.8. Standardized versions of MQV use a hash function such as SHA-1 to build the key derivation function. Indistinguishability is not one of the original goals of SHA-1. It was introduced for the purpose of message digesting in digital signatures, for which purpose it not believed to require indistinguishability. Indistinguishability has been widely adopted as one of the security attributes of SHA-1 and some other hash functions. Indeed, some deterministic random bit generators employ hash functions like SHA-1. So, it may be reasonable to believe that the standardized key derivations for MQV do provide indistinguishability.

Key derivation function d must be non-predicting if it is to be non-distinguishable, because of the following proposition.

Proposition 5.4.3. *If* KDP *is a key derivation predictor for key derivation function* $d : \mathbb{G} \to \mathbb{K}$*, then algorithm* KDD *defined by*

$$\mathsf{KDD}(k) \to k \stackrel{\circ}{=} \mathsf{KDP}() \tag{5.4.10}$$

is a key derivation distinguisher with

- *success rate* $\sigma(\mathsf{KDD}) = \sigma(\mathsf{KDP}) - \frac{2}{|\mathbb{K}|}(1 - \sigma(\mathsf{KDP}))$*, and*
- *computational cost* $\gamma(\mathsf{KDD}) = \gamma(\mathsf{KDP})$.

Proof. Algorithm KDD from (5.4.10) invokes KDP and does a comparison (which we do not count in the cost), so $\gamma(\mathsf{KDD}) = \gamma(\mathsf{KDP})$.

In the general process for KDD (Definition 5.4.2), the value of b computed is $k \stackrel{\circ}{=} \mathsf{KDP}()$. So the value of s computed is

$$(k \stackrel{\circ}{=} \mathsf{KDP}()) \stackrel{\circ}{-} (k \stackrel{\circ}{=} d(U)) \tag{5.4.11}$$

Consider the event that the output KDP() equals $d(U)$. By the Definition 5.4.1, the probability of this event is $\sigma(\mathsf{KDP})$, because U is uniformly distributed in $\mathbb{G}$, as required. In this event, the value of s from (5.4.11) becomes 1, because both sides of the operator are the same.

Otherwise, with probability $1 - \sigma(\mathsf{KDP})$, the complementary event $d(U) \neq \mathsf{KDP}()$ occurs, which we will assume for the remainder of the proof.

The event $k = d(U)$ is independent of the previous event, because k is not given to the algorithm KDP. The probability that $k = d(U)$ is $\frac{1}{2} + \frac{1}{2|\mathbb{K}|}$. In this case, the success indicator for KDD, as given by (5.4.11), evaluates to $(0 \stackrel{\circ}{-} 1) = -1$.

Otherwise the case is that $k \neq d(U)$, which occurs with probability $\frac{1}{2} - \frac{1}{2|\mathbb{K}|}$, which we assume for the rest of the proof.

In this case, the event $k = \mathsf{KDP}()$, occurs with probability $\frac{1}{|\mathbb{K}|-1}$, because k is uniformly distributed in the set $\mathbb{K} \setminus \{d(U)\}$, which contains KDP(), since we have assume $k \neq d(U)$ and $\mathsf{KDP}() \neq d(U)$ above. When $k = \mathsf{KDP}()$ the success indicator for KDD becomes $1 \stackrel{\circ}{-} 0 = -1$.

Otherwise, the $k \neq \mathsf{KDP}()$, which occurs with probability $1 - \frac{1}{|\mathbb{K}|-1}$, and success indicator becomes $0 \stackrel{\circ}{-} 0 = 1$.

Considering all the cases above, gives

$$\sigma(\mathsf{KDD}) = \sigma(\mathsf{KDP}) + (1 - \sigma(\mathsf{KDP}))\left(-\left(\frac{1}{2} + \frac{1}{2|\mathbb{K}|}\right) + \left(\frac{1}{2} - \frac{1}{2|\mathbb{K}|}\right)\left(-\frac{1}{|\mathbb{K}| - 1} + \left(1 - \frac{1}{|\mathbb{K}| - 1}\right)\right)\right) \tag{5.4.12}$$

which simplifies to the claimed success rate. □

Remark 5.4.9. The key derivation distinguisher is the decisional version of the key derivation predictor, which may be regarded as a computational problem.

Remark 5.4.10. Proposition 5.4.3 and its proof about conversion from a computational to a decisional adversary is analogous to Propositions 4.5.3, 4.5.4, and 5.2.6.

5.4.3 Key Derivation Function Seconding Hard

Definition 5.4.4. *An algorithm* KDS *is a key derivation seconder for key derivation function* $d : \mathbb{G} \to \mathbb{K}$ *if it takes as input an element of* $\mathbb{G}$ *and outputs an element from* $\mathbb{G}$*. Its success rate* $\sigma(\mathsf{KDS})$ *is defined as*

the expected value of s in the following process:

$$V \leftarrow \mathbb{G} \tag{5.4.13}$$
$$U \leftarrow \mathsf{KDS}(V) \tag{5.4.14}$$
$$s \leftarrow (d(U) \stackrel{\circ}{=} d(V))(1 - (U \stackrel{\circ}{=} V)) \tag{5.4.15}$$

The hardness condition associated with a key derivation seconder is that a high-success, low-cost KDS is not constructible. This condition can be called seconder resistance (which is similar to the term second pre-image resistance used for hash functions).

Remark 5.4.11. This report has not identified this condition to be necessary for Menezes–Qu–Vanstone to provided any form of security.

Proposition 5.4.5 and Lemma 6.4.1 establishes this condition to be sufficiently-necessary (§6.1) for Menezes–Qu–Vanstone to provide computational passive implicit key authentication..

The key derivation function must be predictor-resistant if it is to be true the key derivation function is second pre-image resistant, because of the following proposition.

Proposition 5.4.5. *If* KDP *is a key derivation function predictor, then the algorithm* KDS *defined by*

$$U \leftarrow \mathbb{G} \tag{5.4.16}$$
$$\mathsf{KDS}(V) \rightarrow U \tag{5.4.17}$$

is a key derivation seconder with

- *success rate* $\sigma(\mathsf{KDS}) \geqslant \sigma(\mathsf{KDP})^2 - \frac{\sigma(\mathsf{KDP})}{n}$.
- *computational cost* $\gamma(\mathsf{KDS}) \leqslant 2\log_2(n)(\gamma(\mathbb{G}) + \gamma(\$))$.

Proof. Algorithm KDS computes one random elements in $\mathbb{G}$, which can be done by selecting to a random elements $u \leftarrow \mathbb{Z}_n$, at cost of at most $2\log_2(n)\gamma(\$)$, and computing $U = uG$ at a cost of at most $2\log_2(n)\gamma(\mathbb{G})$. So, its computational cost $\gamma(\mathsf{KDS})$ has the claimed upper bound.

Let $k \leftarrow \mathsf{KDP}()$. With probability $\sigma(\mathsf{KDP})$, it is true that $d(U) = k$, by the process (Definition 5.4.1) for KDP. Similarly, the probability that $d(V) = k$ is $\sigma(\mathsf{KDP})$. The probability that $U = V$ is $\frac{1}{n}$. So the probability that $d(U) = d(V) = k$ and $U \neq V$ is $\sigma(\mathsf{KDP})(\sigma(\mathsf{KDP}) - \frac{1}{n})$, which gives the desired success rate for KDS. □

Remark 5.4.12. A hard key derivation seconder problem will not be shown to a necessary condition for the security of Menezes–Qu–Vanstone. But if it is hard, then d is predictor-resistant by Proposition 5.4.5, which is a necessary condition (Lemma 6.4.1). We may call such conditions, which imply a necessary condition, sufficiently-necessary.

Remark 5.4.13. In the case of standards for elliptic curve Menezes–Qu–Vanstone, the key derivation function is often based on the x-coordinate of an elliptic point in affine Weierstrass coordinates. In these coordinates, an element U and its negative $-U$ have the same x-coordinate, so $d(U) = d(-U)$.

In this case, we can amend the definition of KDS to define the success indicator as $(d(U) \stackrel{\circ}{=} d(V))(1 - (U \stackrel{\circ}{=} V))(1 - (U \stackrel{\circ}{=} -V))$.

5.4.4 Key Derivation Function Colliding Hard

Definition 5.4.6. *An algorithm* KDC *is a key derivation collider for key derivation function* $d : \mathbb{G} \to \mathbb{K}$ *if it takes no inputs and produces an output of two points in* $\mathbb{G}$*. Its success rate* $\sigma(\mathsf{KDC})$ *is defined by the expected value of* s *in the following process:*

$$(U, V) \leftarrow \mathsf{KDC}() \tag{5.4.18}$$

$$s \leftarrow (d(U) \mathrel{\mathring{=}} d(V))(1 - (U \mathrel{\mathring{=}} V)) \tag{5.4.19}$$

The hardness condition associated with a key derivation function collider is that a high-success, low-cost KDC as above is not constructible. When this condition holds, we may say that d is collider-resistant.

Remark 5.4.14. This report has not identified this condition to be necessary for Menezes–Qu–Vanstone to provided any form of security.

Proposition 5.4.8 and Lemma 6.4.1 establish this condition to be sufficiently-necessary (§6.1) for Menezes–Qu–Vanstone to provide computational passive implicit key authentication.

Remark 5.4.15. When $|\mathbb{K}| > |\mathbb{G}|$, finding a collision is existentially easy, but in many cases, such as when the key derivation is a hash function like SHA-256, appears to be constructibly hard.

In contrast, the seconder problem has the potential to be existentially hard.

Remark 5.4.16. An algorithm KDC_m may be constructed as follows:

$$P_1, \ldots, P_m \leftarrow \mathbb{G} \tag{5.4.20}$$

$$d_i \leftarrow d(P_i) \tag{5.4.21}$$

$$j \leftarrow \mathsf{match}_1(d_1, \ldots, d_m) \tag{5.4.22}$$

$$k \leftarrow \mathsf{match}_2(d_1, \ldots, d_m) \tag{5.4.23}$$

$$\mathsf{KDC}() \to (P_j, P_k) \tag{5.4.24}$$

where (5.4.21) is repeated for i from 1 to m, and where match_1 and match_2 are functions that take m inputs and output the index of the first and second, respectively, of the inputs which equal each other. The exact success rate and cost of this algorithm would be interesting to determine. For $m \approx \sqrt{n}$, this algorithm KDC_m should have fairly high success, because of the well-known birthday surprise phenomenon.

A natural improvement of the algorithm KDC_m would add a check that $P_j \neq P_k$, and searching further through the list of d_i for a collision as needed.

Remark 5.4.17. An algorithm with much lower cost of memory than the algorithm from Remark 5.4.16 for finding collisions is Floyd's cycle-finding algorithm. Pollard's rho algorithm for the discrete logarithm solving is based on the same idea as Floyd's cycle-finding algorithm.

Remark 5.4.18. Because standards for Menezes–Qu–Vanstone key agreement usually specify that the key derivation function d is computed using a simple iterative construction invoking hash function such as SHA-256 that is believed to be constructibly collision-resistant, we can usually infer a similar belief that the key derivation function d is similarly constructibly collider-resistant.

The key derivation function must be second pre-image resistant if it is to be collision resistant, because of the following proposition.

Proposition 5.4.7. *If* KDS *is a key derivation seconder, then the algorithm* KDC *defined by*

$$V \leftarrow \mathbb{G} \tag{5.4.25}$$

$$\mathsf{KDC}() \to (\mathsf{KDS}(V), V) \tag{5.4.26}$$

is a key derivation collider with

- *success rate* $\sigma(\mathsf{KDC}) = \sigma(\mathsf{KDS})$, *and*
- *computational cost* $\gamma(\mathsf{KDC}) \leqslant \gamma(\mathsf{KDS}) + 2\log_2(n)(\gamma(\mathbb{G}) + \gamma(\$))$.

Proof. Algorithm KDC invokes algorithm KDS and selects one random element of $\mathbb{G}$, so $\gamma(\mathsf{KDC})$ has the claimed upper bound.

The input V that KDC provides to KDS is uniformly distributed in $\mathbb{G}$, in accordance with the process for KDS (Definition 5.4.4, because V is uniformly distributed in $\mathbb{G}$ from selection made by the algorithm KDC.

Let U be the output of KDS, when called by KDC. The success indicator for KDC and KDS then have the same formula, so they have the same success rate. □

In other words KDS ⇒ KDC. Proposition 5.4.5 can be summarized as KDP ⇒ KDS. Taking the transitive closure of these results gives KDP ⇒ KDC. If nobody knows how to construct a collider in d, in the sense of Definition 5.4.6, then nobody knows how to exhibit a predictor for d (Definition 5.4.1). As just noted, this follows by taking the transitive closure of Propositions 5.4.5 and 5.4.7, but for convenience, the following direct proposition is stated and proved.

Proposition 5.4.8. *If* KDP *is a key derivation function predictor, then the algorithm* KDC *defined by*

$$U, V \leftarrow \mathbb{G} \tag{5.4.27}$$
$$\mathsf{KDC}() \rightarrow (U, V) \tag{5.4.28}$$

is a key derivation collider with

- *success rate* $\sigma(\mathsf{KDC}) \geqslant \sigma(\mathsf{KDP})^2 - \frac{\sigma(\mathsf{KDP})}{n}$.
- *computational cost* $\gamma(\mathsf{KDC}) \leqslant 4\log_2(n)(\gamma(\mathbb{G}) + \gamma(\$))$.

Proof. Algorithm KDC simply computes two random elements in $\mathbb{G}$, which can be done by selecting to two random elements $u, v \leftarrow \mathbb{Z}_n$, at cost of at most $4\log_2(n)\gamma(\$)$, and computing $U = uG$ and $V = vG$ at a cost of at most $4\log_2(n)\gamma(\mathbb{G})$. So, its computational $\gamma(\mathsf{KDC})$ has the claimed upper bound.

Let $k \leftarrow \mathsf{KDP}()$. With probability $\sigma(\mathsf{KDP})$, it is true that $d(U) = k$, by the process (Definition 5.4.1) for KDP. Similarly, the probability that $d(V) = k$ is $\sigma(\mathsf{KDP})$. The probability that $U = V$ is $\frac{1}{n}$. So the probability that $d(U) = d(V) = k$ and $U \neq V$ is $\sigma(\mathsf{KDP})(\sigma(\mathsf{KDP}) - \frac{1}{n})$, which gives the desired success rate for KDC. □

5.4.5 Key Derivation Function Reversing Hard

Definition 5.4.9. *An algorithm* KDR *is a key derivation reverser for the key derivation function* d *if it takes as input a value in the session key space* $\mathbb{K}$ *and outputs a value in* $\mathbb{G}$. *Its success rate* $\sigma(\mathsf{KDR})$ *is defined as the expected value of* s *in the following process*

$$V \leftarrow \mathbb{G} \tag{5.4.29}$$
$$U \leftarrow \mathsf{KDR}(d(V)) \tag{5.4.30}$$
$$s \leftarrow U \stackrel{?}{=} V \tag{5.4.31}$$

The hardness condition associated with a key derivation reverser is that a high-success, low-cost KDR is not constructible.

Remark 5.4.19. This condition has not been identified in this report as necessary to the security of Menezes–Qu–Vanstone key agreement.

Conjecture 5.5.15 and Lemma 6.6.2 together suggest this condition as partially-necessary (§6.1) for decisional passive implicit key authentication.

If the key space $\mathbb{K}$ is much smaller than the group $\mathbb{G}$, then a high-success key derivation reverser does not exist, as the following proposition shows.

Proposition 5.4.10. *If* KDR *is a key derivation reverser, then* $\sigma(\mathsf{KDR}) \leqslant |\mathbb{K}|/n$.

Proof. Suppose that $|d(\mathbb{G})| = m$. So that d takes on m possible values, say $k_1, \ldots, k_m$. Let $\mathbb{G}_i = d^{-1}(k_i)$, be the pre-image of k_i under d, so $\mathbb{G}_i = \{U : d(U) = k_i\}$. Let $n_i = |\mathbb{G}_i|$. By definition $n_i \geqslant 1$ and $n_1 + \cdots + n_m = n$.

In the process for KDR from Definition 5.4.9, element V is uniformly distributed in $\mathbb{G}$. The event $V \in \mathbb{G}_i$, or equivalently $d(V) = k_i$, has probability n_i/n. There are n_i possible values of U in $\mathbb{G}_i$, and k_i gives KDR no information which is V. So, KDR has probability at most $1/n_i$ of guessing V. Therefore,

$$\sigma(\mathsf{KDR}) \leqslant \sum_{i=1}^{m} \frac{n_i}{n} \frac{1}{n_i}, \tag{5.4.32}$$

which, upon simplification, means that $\sigma(\mathsf{KDR}) \leqslant m/n$. But $m \leqslant |\mathbb{K}|$, so the desired bound holds. □

Furthermore, very-high-success reversers and predictors for a given key derivation function cannot co-exist, because of the following proposition.

Proposition 5.4.11. *If* KDP *is a key derivation predictor for key derivation function* $d : \mathbb{G} \to \mathbb{K}$ *and* KDR *is a key derivation reverser for* d*, then*

$$\sigma(\mathsf{KDP}) + \sigma(\mathsf{KDR}) \leqslant 1 + \frac{1}{n} \tag{5.4.33}$$

where, as usual, $n = |\mathbb{G}|$.

Proof. Suppose that $|d(\mathbb{G})| = m$. So that d takes on m possible values, say $k_1, \ldots, k_m$. Let $\mathbb{G}_i = d^{-1}(k_i)$, be the pre-image of k_i under d, so $\mathbb{G}_i = \{U : d(U) = k_i\}$. Let $n_i = |\mathbb{G}_i|$. By definition $n_i \geqslant 1$ and $n_1 + \cdots + n_m = n$.

In the process for KDR from Definition 5.4.9, element V is uniformly distributed in $\mathbb{G}$. The event $V \in \mathbb{G}_i$, or equivalently $d(V) = k_i$, has probability n_i/n. There are n_i possible values of U in $\mathbb{G}_i$, and k_i gives KDR no information which is V. So, KDR has probability at most $1/n_i$ of guessing V. Therefore,

$$\sigma(\mathsf{KDR}) \leqslant \sum_{i=1}^{m} \frac{n_i}{n} \frac{1}{n_i}, \tag{5.4.34}$$

which, upon simplification, means that $\sigma(\mathsf{KDR}) \leqslant m/n$.

In the process for KDP from Definition 5.4.1, element U is uniformly distributed in $\mathbb{G}$. The success rate of KDP is at most the n_j/n where n_j is the maximal value in the set $\{n_1, \ldots, n_m\}$. Then $n_j = n - \sum_{i \neq j} n_i \leqslant n - (m-1)$. So, $\sigma(\mathsf{KDP}) \leqslant \frac{n-m+1}{n}$.

Summing these two inequalities gives the desired bound. □

Remark 5.4.20. For compatibility with Remark 5.4.13, we can amend Definition 5.4.9 to have success when $U = \pm V$. Formally, $s \leftarrow (U \stackrel{?}{=} V) | (U \stackrel{?}{=} -V)$ where $a|b = a + b - ab$

Remark 5.4.21. Yet another issue with the key derivation reverser is that its success indicator is not determined by solely by its inputs and outputs, but rather by an experiment variable about which only partial information may be supplied as an input to the reverser. (A similar issue applies to a predictor, Remark 5.4.5.)

Algorithms that use a key derivation reverser may rely on this experiment variable, but only with implicit knowledge, and therefore may not be able test the success of a particular run of the reverser.

5.4.6 Key Derivation Function Opening Hard

Definition 5.4.12. *An algorithm* KDO *is a key derivation opener for the key derivation function d if it takes as input a value in the session key space $\mathbb{K}$ and outputs a value in $\mathbb{G}$. Its success rate $\sigma(\mathsf{KDO})$ is defined as the expected value of s in the following process*

$$V \leftarrow \mathbb{G} \tag{5.4.35}$$
$$U \leftarrow \mathsf{KDO}(d(V)) \tag{5.4.36}$$
$$s \leftarrow d(U) \stackrel{?}{=} d(V) \tag{5.4.37}$$

The hardness condition associated with a key derivation opener is that a high-success low-cost KDO is not constructible.

Remark 5.4.22. This condition has not been identified in this report as necessary to the security of Menezes–Qu–Vanstone key agreement.

Remark 5.4.23. One specific algorithm KDO works as follows. First, a one-time setup procedure is executed (before receiving its input key k) evaluate d on every element V of $\mathbb{G}$, in some uniformly random order, and store the pairs $(d(V), V)$ in a table, perhaps sorted by value of $d(V)$. Second, upon receiving k, search for an appearance (k, U) in the table, using say binary search, and then output a value of U.

The naive implementation of this algorithm uses a large amount memory, at least $n(\log_2(|\mathbb{K}|) + \log_2(n))$, and a large one-time set-up cost of at least $n\gamma(\mathbb{G})$. The run-time cost may be fairly small. The success rate of this algorithm is 1.

Reversing a key derivation function must be hard if opening a key derivation function is hard, because of the following proposition.

Proposition 5.4.13. *If* KDR *is a key derivation reverser for key derivation function $d : \mathbb{G} \to \mathbb{K}$, then the algorithm* KDO *defined by*

$$\mathsf{KDO}(k) \to \mathsf{KDR}(k) \tag{5.4.38}$$

is a key derivation opener for d with

- *success rate $\sigma(\mathsf{KDO}) \geqslant \sigma(\mathsf{KDR})$, and*
- *computational cost $\gamma(\mathsf{KDO}) = \gamma(\mathsf{KDR})$.*

Proof. Algorithm KDO above consists of invoking algorithm KDR, so its computational cost is $\gamma(\mathsf{KDO}) = \gamma(\mathsf{KDR})$.

The input k that KDO provides to KDR is in accordance with the distribution of the input for KDR according to its process (Definition 5.4.9), because KDO has the same distribution in its process (Definition 5.4.12). So, with probability $\sigma(\mathsf{KDR})$, the success indicator of KDR has value 1, which means that $U = V$, where U is the output of KDR.

In this case, $d(U) = d(V)$, and KDO has success indicator 1 too. So, $\sigma(\mathsf{KDO}) \geqslant \sigma(\mathsf{KDR})$, as claimed. □

Proposition 5.4.13 may be summarized as $\mathsf{KDR} \Rightarrow \mathsf{KDO}$. As partial converse is the following proposition.

Proposition 5.4.14. *If* KDO *is a key derivation opener for* $d : \mathbb{G} \to \mathbb{K}$*, then algorithm* KDR *defined by*

$$\mathsf{KDR}(k) \to \mathsf{KDO}(k) \tag{5.4.39}$$

is a key derivation reverser for d*, and algorithm* KDS *defined by*

$$\mathsf{KDS}(V) \to \mathsf{KDO}(d(V)) \tag{5.4.40}$$

is a key derivation seconder for d*, with*

- *success rates such that* $\sigma(\mathsf{KDR}) + \sigma(\mathsf{KDS}) \geqslant \sigma(\mathsf{KDO})$*, and*
- *computational costs* $\gamma(\mathsf{KDR}) = \gamma(\mathsf{KDO})$ *and* $\gamma(\mathsf{KDS}) = \gamma(\mathsf{KDO}) + \gamma(d)$.

Proof. Algorithm KDR from (5.4.39) just invokes algorithm KDO, so its computational cost $\gamma(\mathsf{KDR}) = \gamma(\mathsf{KDO})$. Algorithm KDS from (5.4.40) invokes algorithm KDO and evaluates d, so its computational cost $\gamma(\mathsf{KDS})$ has the claimed upper bound.

The process for KDR from Definition 5.4.9 selects V uniformly at random from $\mathbb{G}$, computes $k = d(V)$, which is the given as input to KDR. Algorithm KDR from (5.4.39) feeds this k to KDO, which according Definition 5.4.12, is expecting k with the same distribution. So, with probability $\sigma(\mathsf{KDO})$, algorithm KDO has success indicator 1, which means that $d(U) = d(V)$, where U is the output of KDO.

The process for KDS from Definition 5.4.4 selects V uniformly at random from $\mathbb{G}$. Algorithm KDS from (5.4.40) computes $k = d(V)$, and feeds k to KDO, which according to Definition 5.4.12, is expecting k with such a distribution. So, , with probability $\sigma(\mathsf{KDO})$, algorithm KDO has success indicator 1, which means that $d(U) = d(V)$, where U is the output of KDO.

From the perspective of KDO its role in KDS or KDR is the same. Consider the two cases:.

1. If $U = V$, then KDR would have success indicator 1.
2. If $U \neq V$, then KDS would have success indicator 1.

The probabilities of each case is the same in its role as KDR or KDS. The sums of the probability of the case is 1 (after accounting for the event $d(U) = d(V)$, which occurred with probability $\sigma(\mathsf{KDO})$). Therefore, $\sigma(\mathsf{KDR}) + \sigma(\mathsf{KDS}) \geqslant \sigma(\mathsf{KDO})$. □

We may summarize this result as $\mathsf{KDO} \Rightarrow (\mathsf{KDR}|\mathsf{KDS})$. So, if key derivation seconding is hard, then key derivation opening is equivalent to key derivation reversing.

5.4.7 Key Derivation Function Inverting Hard

Definition 5.4.15. *An algorithm* KDI *is a key derivation inverter for the key derivation function* d *if it takes as input a value in the session key space* $\mathbb{K}$ *and outputs a value in* $\mathbb{G}$*. Its success rate* $\sigma(\mathsf{KDI})$ *is defined as the expected value of* s *in the following process*

$$k \leftarrow \mathbb{K} \tag{5.4.41}$$
$$U \leftarrow \mathsf{KDI}(k) \tag{5.4.42}$$
$$s \leftarrow k \stackrel{?}{=} d(U) \tag{5.4.43}$$

The hardness condition associated with key derivation inverter is that a high-success low-cost KDI is not constructible.

Proposition 5.4.16. *If* KDI *is a key derivation inverter, then* $\sigma(\mathsf{KDI}) \leqslant n/|\mathbb{K}|$.

Proof. Let $m = |d(\mathbb{G})|$. As $k \leftarrow \mathbb{K}$ in the process for KDI (Definition 5.4.15), the probability that $k \in d(\mathbb{G})$ is $m/|\mathbb{K}|$. If $k \notin d(\mathbb{G})$, then there does not exist $U \in \mathbb{G}$ such that $k = d(U)$, and KDI will have success indicator 0. Therefore $\sigma(\mathsf{KDI}) \leqslant m/|\mathbb{K}|$. But $m \leqslant n$, so the claimed bound holds. □

Remark 5.4.24. Proposition 5.4.10 bounds $\sigma(\mathsf{KDR})$ above, but with the reciprocal of the bound from Proposition 5.4.16.

If key derivation distinguishing (Definition 5.4.2) is hard and $|\mathbb{K}|$ is large, then key derivation inverting is equivalent to key derivation opening (Definition 5.4.12), because of the following two similar propositions.

Proposition 5.4.17. *If* KDO *is a key derivation opener for* d*, then algorithm* KDI *defined by*

$$\mathsf{KDI}(k) \to \mathsf{KDO}(k) \tag{5.4.44}$$

is a key derivation inverter for d*, and algorithm* KDD *defined by*

$$\mathsf{KDD}(k) \to k \stackrel{\circ}{=} d(\mathsf{KDO}(k)) \tag{5.4.45}$$

is a key derivation distinguisher for d*, with*

- *success rates such that* $\sigma(\mathsf{KDI})\left(1 - \frac{2}{|\mathbb{K}|}\right) + \sigma(\mathsf{KDD}) = \sigma(\mathsf{KDO}) - \frac{1}{|\mathbb{K}|}$,
- *and computational costs* $\gamma(\mathsf{KDI}) = \gamma(\mathsf{KDO})$ *and* $\gamma(\mathsf{KDD}) \leqslant \gamma(\mathsf{KDO}) + \gamma(d)$.

Proof. Algorithm KDI from (5.4.44) consists of invoking algorithm KDO, so its computational cost is $\gamma(\mathsf{KDI}) = \gamma(\mathsf{KDO})$. Algorithm KDD from (5.4.45) invokes algorithm KDO evaluates function d, and does a comparison (which we do not count in the cost), so its computational cost is $\gamma(\mathsf{KDD}) \leqslant \gamma(\mathsf{KDO}) + \gamma(d)$.

Under the process for KDD (Definition 5.4.2), session key k is selected according to the split distribution $\mathbb{K} \vee d(U)$, where U is selected uniformly at random from $\mathbb{G}$. The success rate $\sigma(\mathsf{KDD})$ is therefore the average value of the success rates of KDD under the two separate processes, one process where $k \leftarrow \mathbb{K}$ and the other process where $k \leftarrow d(U)$. Let these success rates be σ_0 and σ_1, respectively. So $\sigma(\mathsf{KDD}) = \frac{\sigma_0 + \sigma_1}{2}$.

In the algorithm KDD from (5.4.45), the success indicator evaluates to

$$(k \stackrel{\circ}{=} d(\mathsf{KDO}(k))) \stackrel{\circ}{-} (k \stackrel{\circ}{=} d(U)) \tag{5.4.46}$$

Under the process with $k \leftarrow d(U)$, then the input to KDO is distributed in accordance with process for KDO (Definition 5.4.12). So, with probability $\sigma(\mathsf{KDO})$, the event that algorithm KDO has success indicator 1 occurs, which means that $d(\mathsf{KDO}(k)) = d(U)$ occurs. In this case, the two sides of the operator $\stackrel{\circ}{-}$ in (5.4.46) are equal, so the success indicator for KDD evaluates to one. Failure of this event means $d(\mathsf{KDO}(k)) \neq d(U)$ and occurs with probability $1 - \sigma(\mathsf{KDO})$. In this case, the success indicator for KDD evaluates to $(0 \stackrel{\circ}{-} 1) = -1$. Therefore, $\sigma_1 = \sigma(\mathsf{KDO}) - (1 - \sigma(\mathsf{KDO}))$.

The invocation of KDO in (5.4.46) (as the success indicator of our algorithm KDD) can regarded as an invocation of algorithm KDI from (5.4.44). Under the process with $k \leftarrow \mathbb{K}$, then the input k is distributed in accordance with the process for KDI (Definition 5.4.15). So, with probability $\sigma(\mathsf{KDI})$, whatever that value may be (not necessarily a priori related to the the success rate of $\sigma(\mathsf{KDO})$), the success indicator

for KDI is 1. Let V be the output of KDI (and hence KDO). A success indicator 1 for KDI, means that $k = d(V)$.

In the event of the success indicator 1 for KDI, the success indicator for KDD, as given by (5.4.46), simplifies to $1 \div (k \doteq d(U))$ because the left side of the operator $\div$ becomes 1. The right side of the operator equals 1 with probability $\frac{1}{|\mathbb{K}|}$, and is otherwise 0, because $k \leftarrow K$, and k is independent of U. Therefore, the expected value of success indicator in this case is $\frac{1}{|\mathbb{K}|} - \frac{|\mathbb{K}|-1}{|\mathbb{K}|} = -1 + \frac{2}{|\mathbb{K}|}$.

Otherwise, KDI has success indicator 0, which occurs with probability $1 - \sigma(\mathsf{KDI})$, and means that $k \neq d(V)$. This causes the left side of the operator $\div$ in the success indicator for KDD, as given by (5.4.46), to evaluate to 0. The right side remains distributed as above (in the case where KDI succeeds), and is a value in $\{0, 1\}$. but $(0 \div r) = -(1 \div r)$ for $r \in \{0, 1\}$, so the expected value of the KDD success indicator when KDI fails is the negative of the value above.

Therefore, $\sigma_0 = (\sigma(\mathsf{KDI}) - (1 - \sigma(\mathsf{KDI}))(-1 + \frac{2}{|\mathbb{K}|})$. Taking the average of σ_0 and σ_1 gives

$$\begin{aligned} \sigma(\mathsf{KDD}) &= \frac{1}{2}(2\sigma(\mathsf{KDO}) - 1) + \frac{1}{2}(2\sigma(\mathsf{KDI}) - 1)\left(-1 + \frac{2}{|\mathbb{K}|}\right) \\ &= \sigma(\mathsf{KDO}) - \sigma(\mathsf{KDI}) - \frac{1}{|\mathbb{K}|} + \frac{2\sigma(\mathsf{KDI})}{|\mathbb{K}|} \end{aligned} \tag{5.4.47}$$

which implies the claimed relation between success rates. □

Proposition 5.4.18. *If* KDI *is a key derivation inverter for d, then algorithm* KDO *defined by*

$$\mathsf{KDO}(k) \to \mathsf{KDI}(k) \tag{5.4.48}$$

is a key derivation inverter for d, and algorithm KDD *defined by*

$$\mathsf{KDD}(k) \to 1 - (k \doteq d(\mathsf{KDI}(k))) \tag{5.4.49}$$

is a key derivation distinguisher for d, with

- *success rates such that* $\sigma(\mathsf{KDO}) + \sigma(\mathsf{KDD}) > \sigma(\mathsf{KDI}) - \frac{2\sigma(\mathsf{KDI})}{|\mathbb{K}|}$,
- *and computational costs* $\gamma(\mathsf{KDO}) = \gamma(\mathsf{KDI})$ *and* $\gamma(\mathsf{KDD}) \leqslant \gamma(\mathsf{KDI}) + \gamma(d)$.

Proof. Algorithm KDO from (5.4.48) consists of invoking algorithm KDI, so its computational cost is $\gamma(\mathsf{KDO}) = \gamma(\mathsf{KDI})$. Algorithm KDD from (5.4.49) invokes algorithm KDI evaluates function d, does a comparison (which we do not count in the cost), and does a bit subtraction (which we do not count in the cost), so its computational cost is $\gamma(\mathsf{KDD}) \leqslant \gamma(\mathsf{KDI}) + \gamma(d)$.

Under the process for KDD (Definition 5.4.2), session key k is selected according to the split distribution $\mathbb{K} \vee d(U)$, where U is selected uniformly at random from $\mathbb{G}$. The success rate $\sigma(\mathsf{KDD})$ is therefore the average value of the success rates of KDD under the two separate processes, one process where $k \leftarrow \mathbb{K}$ and the other process where $k \leftarrow d(U)$. Let these success rates be σ_0 and σ_1, respectively. So $\sigma(\mathsf{KDD}) = \frac{\sigma_0 + \sigma_1}{2}$.

In the algorithm KDD from (5.4.49), the success indicator, per Definition 5.4.2, evaluates to

$$(1 - (k \doteq d(\mathsf{KDI}(k)))) \div (k \doteq d(U)) \tag{5.4.50}$$

The invocation of KDO in (5.4.50) (as the success indicator of our algorithm KDD) can regarded as an invocation of algorithm KDI from (5.4.48). Under the process with $k \leftarrow d(U)$, then the input to KDO is

distributed in accordance with process for KDO (Definition 5.4.12). So, with probability $\sigma(\mathsf{KDO})$, the event that algorithm KDO has success indicator 1 occurs, which means that $d(\mathsf{KDO}(k)) = d(U)$ occurs. In this case, the two sides of the operator $\div$ in (5.4.46) are unequal, so the success indicator for KDD evaluates to -1. Failure of KDO, meaning $d(\mathsf{KDO}(k)) \neq d(U)$, occurs with probability $1 - \sigma(\mathsf{KDO})$. In this case, the success indicator for KDD evaluates to $((1 - 0) \div 1) = 1$. Therefore, $\sigma_1 = -\sigma(\mathsf{KDO}) + (1 - \sigma(\mathsf{KDO}))$.

Under the process with $k \leftarrow \mathbb{K}$, then the input k is distributed in accordance with the process for KDI (Definition 5.4.15). So, with probability $\sigma(\mathsf{KDI})$, whatever that value may be (not necessarily a priori related to the the success rate of $\sigma(\mathsf{KDO})$), the success indicator for KDI is 1. Let V be the output of KDI (and hence KDO). A success indicator 1 for KDI, means that $k = d(V)$.

In the event of the success indicator 1 for KDI, the success indicator for KDD, as given by (5.4.50), simplifies to $0 \div (k \stackrel{\circ}{=} d(U))$ because the left side of the operator $\div$ becomes $1 - 1 = 0$. The right side of the operator equals 1 with probability $\frac{1}{|\mathbb{K}|}$, and is otherwise 0, because $k \leftarrow K$, and k is independent of U. Therefore, the expected value of success indicator in this case is $-\frac{1}{|\mathbb{K}|} + \frac{|\mathbb{K}|-1}{|\mathbb{K}|} = 1 - \frac{2}{|\mathbb{K}|}$.

Otherwise, KDI has success indicator 0, which occurs with probability $1 - \sigma(\mathsf{KDI})$, and means that $k \neq d(V)$. This cause the left side of the operator $\div$ in the success indicator for KDD, as given by (5.4.46), to evaluate to $1 = 1 - 0$. The right side remains distributed as above (in the case where KDI succeeds), and is value in $\{0, 1\}$. but $(1 \div r) = -(0 \div r)$ for $r \in \{0, 1\}$, so the expected value of the KDD success indicator when KDI fails is the negative of the value above.

Therefore, $\sigma_0 = (-\sigma(\mathsf{KDI}) + (1 - \sigma(\mathsf{KDI}))(-1 + \frac{2}{|\mathbb{K}|})$. Taking the average of σ_0 and σ_1 gives

$$\begin{aligned} \sigma(\mathsf{KDD}) &= \frac{1}{2}(1 - 2\sigma(\mathsf{KDO})) + \frac{1}{2}(2\sigma(\mathsf{KDI}) - 1)\left(1 - \frac{2}{|\mathbb{K}|}\right) \\ &= -\sigma(\mathsf{KDO}) + \sigma(\mathsf{KDI}) + \frac{1}{|\mathbb{K}|} - \frac{2\sigma(\mathsf{KDI})}{|\mathbb{K}|} \end{aligned} \tag{5.4.51}$$

which implies the claimed bound on $\sigma(\mathsf{KDD}) + \sigma(\mathsf{KDO})$. □

5.5 Joint Conditions on the Group and Key Derivation Function

5.5.1 Derived Diffie–Hellman Problem Hard

Definition 5.5.1. *An algorithm* KDH *is a derived Diffie–Hellman problem solver if it takes as input two elements of* $\mathbb{G}$ *and outputs an element of the key space* $\mathbb{K}$*. It success rate* $\sigma(\mathsf{KDH})$ *is defined to be the expected value of* s *in the following process:*

$$P, Q \leftarrow \mathbb{G} \tag{5.5.1}$$
$$k \leftarrow \mathsf{KDH}(P, Q) \tag{5.5.2}$$
$$s \leftarrow k \stackrel{\circ}{=} d(P \otimes Q) \tag{5.5.3}$$

The hardness condition associated with derived Diffie–Hellman problem is that no high-success, low-cost KDH is constructible.

Remark 5.5.1. Hardness of the derived Diffie–Hellman problem is necessary condition for the Menezes–Qu–Vanstone key agreement to provide computational passive implicit key authentication because of Lemma 6.6.1.

Remark 5.5.2. Hardness of the derived Diffie–Hellman problem is a sufficient condition for the Menezes–Qu–Vanstone key agreement to provide computational passive implicit key authentication because of Theorem 7.2.1.

Both the Diffie–Hellman problem must be hard and the key derivation function must be non-predicting if it is to be true that the derived Diffie–Hellman problem is hard, because of the following two lemmas.

Proposition 5.5.2. *If* DH *is a Diffie–Hellman problem solver, then algorithm* KDH *defined by*

$$\mathsf{KDH}(P, Q) \to d(\mathsf{DH}(P, Q)) \tag{5.5.4}$$

is a derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{KDH}) \geqslant \sigma(\mathsf{DH})$, *and*
- *computational cost* $\gamma(\mathsf{KDH}) \leqslant \gamma(\mathsf{DH}) + \gamma(d)$.

Proof. Algorithm KDH from (5.5.4) above invokes algorithm DH and evaluates function d, so its computational cost is $\gamma(\mathsf{KDH}) \leqslant \gamma(\mathsf{DH}) + \gamma(d)$.

The process (Definition 5.2.3) for DH has the same distribution of inputs as the process (Definition 5.5.1) for KDH. Therefore, DH has probability $\sigma(\mathsf{DH})$ of having its success indicator evaluate to 1.

The success indicator of DH is $\mathsf{DH}(P, Q) \mathrel{\mathring{=}} P \otimes Q$. The success indicator of KDH from (5.5.4) above is obtained from the success indicator of DH by applying d to both sides of the $\mathrel{\mathring{=}}$ operator, which cannot decrease its value. In particular, $\sigma(\mathsf{KDH}) \geqslant \sigma(\mathsf{DH})$. □

Proposition 5.5.3. *If* KDP *is a key derivation predictor, then algorithm* KDH *defined by*

$$\mathsf{KDH}(P, Q) \to \mathsf{KDP}() \tag{5.5.5}$$

is a derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{KDH}) \geqslant \sigma(\mathsf{KDP})\left(1 - \frac{1}{n}\right)$, *and*
- *computational cost* $\gamma(\mathsf{KDH}) = \gamma(\mathsf{KDP})$.

Proof. Algorithm KDH from (5.5.5) above consists of invoking algorithm KDP, so its computational cost is $\gamma(\mathsf{KDH}) = \gamma(\mathsf{KDP})$.

Consider $U = P \otimes Q$. The distribution of U can be regarded as $U \leftarrow 0 \vee_n \mathbb{G}$, where $\vee_n$ is the like the operator $\vee$ but chooses the left distribution with probability $1/n$, and otherwise chooses the right distribution.

So, we may say that, in the process for KDH (Definition 5.5.1), with probability $\frac{n-1}{n}$, element U is distributed uniformly in $\mathbb{G}$, in accordance with process for KDP (Definition 5.4.1). So, in this case, KDP succeeds with probability $\sigma(\mathsf{KDP})$.

When KDP succeeds, we have $k = d(U)$, which yields success for KDH too. □

These two results may be summarized as

$$(\mathsf{DH}|\mathsf{KDP}) \Rightarrow \mathsf{KDH}, \tag{5.5.6}$$

A partial converse to the result $\mathsf{DH} \Rightarrow \mathsf{KDH}$ from Proposition 5.5.2 is the next proposition, which may be summarized as

$$(\mathsf{KDH}\&\mathsf{KDR}) \Rightarrow \mathsf{DH}. \tag{5.5.7}$$

Proposition 5.5.4. *If* KDH *is a derived Diffie–Hellman problem solver and* KDR *is a key derivation function reverser, then the algorithm* DH *defined by*

$$\mathsf{DH}(P, Q) \to \mathsf{KDR}(\mathsf{KDH}(P, Q)) \tag{5.5.8}$$

is Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{DH}) \geqslant \big(\sigma(\mathsf{KDR}) + \sigma(\mathsf{KDH}) - 1 - \frac{1}{n}\big)$, *and*
- *computational cost* $\gamma(\mathsf{DH}) = \gamma(\mathsf{KDR}) + \gamma(\mathsf{KDH})$.

Proof. Algorithm DH invokes KDH and KDR, so its computational cost is $\gamma(\mathsf{DH})$ is as claimed.

Let $V = P \otimes Q$. The distribution of V can be regarded as $0 \vee_n \mathbb{G}$, where $\vee_n$ is the like the operator $\vee$ except that $\vee_n$ chooses the left distribution with probability $1/n$, and otherwise chooses the right distribution. In the event that the right distribution is chosen, which occurs with probability $\frac{n-1}{n}$, element V is distributed uniformly in $\mathbb{G}$.

Let k be the output of $\mathsf{KDH}(P, Q)$ in the algorithm DH from (5.5.8) above. Let R be the output of KDR in the algorithm DH. The success indicator for DH, according to Definition 5.2.3, is

$$R \stackrel{\circ}{=} P \otimes Q \tag{5.5.9}$$

The inputs P and Q that DH provides to KDH are independently and uniformly distributed in $\mathbb{G}$, in accordance with the process for KDH (Definition 5.5.1) because under the process for DH (Definition 5.2.3), they are also so distributed. So, with probability $\sigma(\mathsf{KDH})$, the output of KDH is $k = d(P \otimes Q)$.

With probability $\sigma(\mathsf{KDR})$, algorithm KDR has success indicator 1. If KDH was successful and $\vee_n$ chose V uniformly, the input that DH provides to KDR is $d(P \otimes Q) = d(V)$ and V is a uniformly distributed element of $\mathbb{G}$, in accordance with the process of KDR (Definition 5.4.9), which means that its output $U = R$ is such that $R = V = P \otimes Q$. So, this gives success indicator of 1 for $\sigma(\mathsf{DH})$.

For three events of probabilities p_1 and p_2 and p_3, the probability that all three occur is at least $p_1 + p_2 + p_3 - 2$. In particular, the probability of the three events that KDH succeeds, KDR succeeds, and V is distributed uniformly occur with probability occurs with probability at least $\sigma(\mathsf{KDH}) + \sigma(\mathsf{KDR}) - 1 - \frac{1}{n}$. □

Remark 5.5.3. In the proof, the success rates of KDR and KDH are not necessarily independent, so they cannot be more simply multiplied.

Remark 5.5.4. The bound on $\sigma(\mathsf{DH})$ in Proposition 5.5.4 can be negative, in which case $\sigma(\mathsf{DH}) \geqslant 0$ is a stronger bound, and the proposition provides essentially nothing: it produces a vacuous bound.

Two essentially opposite interpretations of Proposition 5.5.4 are possible, as follows.

1. If the Diffie–Hellman problem is hard, then (a) the derived Diffie–Hellman problem is hard, or (b) key derivation reversing is hard. The latter conclusion does not appear to follow because the Diffie–Hellman problem is not directly related to the key derivation function. So, it seems a reasonable interpretation of this result to say that, *if the Diffie–Hellman problem is hard, then so is the derived Diffie–Hellman problem.*

2. On the other hand, if the key derivation function is indeed hard to reverse—which is what we tend to believe for standardized key derivation function—then Proposition 5.5.4 above does not rule out the possibility that the derived Diffie–Hellman problem is, somehow, easy, while the Diffie–Hellman problem is hard.

5.5.2 Reversibly-Derived Diffie–Hellman Problem Hard

Definition 5.5.5. *An algorithm* RKD *is a reversibly-derived Diffie–Hellman problem if it takes as input two elements in* $\mathbb{G}$ *and an oracle key derivation reverser of given success rate* $\sigma(\mathsf{KDR})$*, and outputs an element of* $\mathbb{G}$*. Its success rate is defined by the expected value of* s *in the following process:*

$$P, Q \leftarrow \mathbb{G} \tag{5.5.10}$$

$$R \leftarrow \mathsf{RKD}(P, Q, \mathsf{KDR}) \tag{5.5.11}$$

$$s \leftarrow R \stackrel{?}{=} P \otimes Q \tag{5.5.12}$$

The hardness condition associated with a reversibly-derived Diffie–Hellman problem solver is a high-success, low-cost RKD is not constructible.

Remark 5.5.5. A hard reversibly-derived Diffie–Hellman problem is sufficient for Menezes–Qu–Vanstone key agreement to provide computational passive key agreement, because of Theorem 7.2.1 and Proposition 5.5.7.

Remark 5.5.6. A summarized statement of the problem is $\mathsf{RKD} = (\mathsf{KDR} \Rightarrow \mathsf{DH})$.

Remark 5.5.7. Proposition 5.4.10 limits the success rate of a key derivation reverser. An oracle key derivation reverser with greater success rate is an impossibility, and the reversibly-derived Diffie–Hellman problem with such an impossibly successful oracle is a nonsensical problem. Therefore, the reversibly-derived Diffie–Hellman problem is quizzical problem.

Remark 5.5.8. If RKD is a reversibly-derived Diffie–Hellman problem solver of purported success rate $\sigma(\mathsf{RKD})$ and key derivation reversing is hard, either because it is impossible (in the sense that Proposition 5.4.10 limits the success rate) or the key derivation opening problem appears hard (Proposition 5.4.13 then ensures that reversing is hard), then it is not immediately clear how to test the purported success rate of RKD, because providing its required oracle is too hard.

If the reversibly-derived Diffie–Hellman problem is to be hard, then the Diffie–Hellman problem must be hard, because of the following proposition.

Proposition 5.5.6. *If* DH *is a Diffie–Hellman problem solver, then algorithm* RKD *defined by*

$$\mathsf{RKD}(P, Q, \mathsf{KDR}) \to \mathsf{DH}(P, Q) \tag{5.5.13}$$

is a reversibly-derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{RKD}) = \sigma(\mathsf{DH})$*, and*
- *computational cost* $\gamma(\mathsf{RKD}) = \gamma(\mathsf{DH})$.

Proof. Algorithm RKD from (5.5.13) consists of invoking DH, so its computational is $\gamma(\mathsf{RKD}) = \gamma(\mathsf{DH})$, as claimed.

The process for RKD (Definition 5.5.5) has the same distribution on inputs P, Q as the process for DH (Definition 5.2.3); and it has the same success indicator formula. Therefore $\sigma(\mathsf{RKD}) = \sigma(\mathsf{DH})$, as claimed. □

Remark 5.5.9. When $\mathbb{K} = \mathbb{G}$ and the key derivation function has $d(U) = U$, then the oracle KDR is easily evaluated, so the RKD becomes equivalent to DH.

Remark 5.5.10. When $\mathbb{K} = \mathbb{G}$ and $g : \mathbb{G} \to \mathbb{Z}_n$ is some arbitrary bijection, such that g and g^{-1} are efficiently computable, and the key derivation function is $d(U) = \exp_G(g(U))$, then the oracle KDR is essentially an oracle for DL, so Proposition 5.2.4 can be used, and the problem RKD becomes easy.

Remark 5.5.11. For arbitrary key derivation functions, such as those based on SHA-1 as in standardized versions of Menezes–Qu–Vanstone key agreement, it would be very surprising if a key derivation reverser could help one to solve the Diffie–Hellman problem. Remark 5.5.10 shows that theoretical helpful reversers exist, but it would seem unlikely that SHA-1 could be used to solve discrete logarithms.

So, it seems reasonable to conjecture that, for a typical key derivation function, the reversibly-derived Diffie–Hellman problem is essentially equivalent to the Diffie–Hellman problem.

If the derived Diffie–Hellman problem is to be hard, then the reversibly-derived Diffie–Hellman problem must be hard, because of the following proposition.

Proposition 5.5.7. *If* KDH *is a derived Diffie–Hellman problem solver, then algorithm* RKD *defined by*

$$\mathsf{RKD}(P, Q, \mathsf{KDR}) \to \mathsf{KDR}(\mathsf{KDH}(P, Q)) \tag{5.5.14}$$

is a reversibly-derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{RKD}) \geqslant (\sigma(\mathsf{KDH}) + \sigma(\mathsf{KDR}) - 1 - \frac{1}{n})$, *and*
- *computational cost* $\gamma(\mathsf{RKD}) = \gamma(\mathsf{KDH})$.

Proof. Algorithm RKD from (5.5.14) above consists of invoking KDH, and calling its oracle KDR (whose cost we do not count in the cost of RKD), so its computational cost is $\gamma(\mathsf{RKD}) = \gamma(\mathsf{KDH})$, as claimed.

The success rate is the same as the success rate from Proposition 5.5.4, and is proved similarly. □

Because $\mathsf{KDP} \Rightarrow \mathsf{KDH}$ from Proposition 5.5.3 and $\mathsf{KDH} \Rightarrow \mathsf{RKD}$ from Proposition 5.5.7, it should follow that $\mathsf{KDP} \Rightarrow \mathsf{RKD}$. We formally state this directly, for convenience.

Proposition 5.5.8. *If* KDP *is a key derivation predictor, then algorithm* RKD *defined by*

$$\mathsf{RKD}(P, Q, \mathsf{KDR}) \to \mathsf{KDR}(\mathsf{KDP}()) \tag{5.5.15}$$

is a reversibly-derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{RKD}) \geqslant (\sigma(\mathsf{KDP}) + \sigma(\mathsf{KDR}) - 1 - \frac{1}{n})$,
- *computational cost* $\gamma(\mathsf{RKD}) = \gamma(\mathsf{KDP})$.

Proposition 5.4.11 implies $\sigma(\mathsf{KDP}) + \sigma(\mathsf{KDR}) - 1 \leqslant \frac{1}{n}$, so the lower bound on $\sigma(\mathsf{RKD})$ in Proposition 5.5.8 is just 0. Therefore, even though implications $\mathsf{KDP} \Rightarrow \mathsf{KDH}$ and $\mathsf{KDH} \Rightarrow \mathsf{RKD}$ both preserve high success rates, their transitive composition $\mathsf{KDP} \Rightarrow \mathsf{RKD}$ does not preserve high success rates. This lack of transitivity appears to be due to the unusual oracle-dependency of the problem RKD, the oracle being incompatible with KDP.

5.5.3 Openably-Derived Diffie–Hellman Problem Hard

Definition 5.5.9. *An algorithm* OKD *is an openably-derived Diffie–Hellman problem solver if it takes as input two elements in* $\mathbb{G}$ *and an oracle key derivation opener of given success rate* $\sigma(\mathsf{KDO})$*, and outputs an element of* $\mathbb{G}$*. Its success rate is defined by the expected value of* s *in the following process:*

$$P, Q \leftarrow \mathbb{G} \tag{5.5.16}$$
$$R \leftarrow \mathsf{OKD}(P, Q, \mathsf{KDO}) \tag{5.5.17}$$
$$s \leftarrow R \stackrel{?}{=} P \otimes Q \tag{5.5.18}$$

The hardness condition associated with an openably-derived Diffie–Hellman problem solver is a high-success, low-cost OKD is not constructible.

Remark 5.5.12. A hard openably-derived Diffie–Hellman problem is partially-sufficient for Menezes–Qu–Vanstone key agreement to provide computational passive key agreement, because of Theorem 7.2.1 and Proposition 5.5.11.

Remark 5.5.13. A summarized statement of the problem is $\mathsf{OKD} = (\mathsf{KDO} \Rightarrow \mathsf{DH})$.

Remark 5.5.14. Because of the lack of an information-theoretic barrier of the success rate of KDO oracle, the openably-derived Diffie–Hellman problem is not a quizzical problem, unlike the reversibly-derived Diffie–Hellman problem, as noted in Remark 5.5.7. It can thus be hoped that transitivity of $\Rightarrow$ applies.

If the openably-derived Diffie–Hellman problem is to be hard, then the Diffie–Hellman problem must be hard, because of the following proposition.

Proposition 5.5.10. *If* DH *is a Diffie–Hellman problem solver, then algorithm* OKD *defined by*

$$\mathsf{OKD}(P, Q, \mathsf{KDO}) \to \mathsf{DH}(P, Q) \tag{5.5.19}$$

is an openably-derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{OKD}) = \sigma(\mathsf{DH})$*, and*
- *computational cost* $\gamma(\mathsf{OKD}) = \gamma(\mathsf{DH})$.

Proof. Algorithm OKD from (5.5.19) consists of invoking DH, so its computational is $\gamma(\mathsf{OKD}) = \gamma(\mathsf{DH})$, as claimed.

The process for OKD (Definition 5.5.9) has the same distribution on inputs P, Q as the process for DH (Definition 5.2.3); and it has the same success indicator formula. Therefore $\sigma(\mathsf{OKD}) = \sigma(\mathsf{DH})$, as claimed. □

Remark 5.5.15. When $\mathbb{K} = \mathbb{G}$ and the key derivation function is defined as $d(U) = U$ for all $U \in \mathbb{G}$, then the oracle KDO is easily evaluated, so the OKD becomes equivalent to DH.

Remark 5.5.16. When $\mathbb{K} = \mathbb{G}$ and $g : \mathbb{G} \to \mathbb{Z}_n$ is some arbitrary bijection, such that g and g^{-1} are efficiently computable, and the key derivation function is $d(U) = \exp_G(g(U))$, then the oracle KDO is essentially an oracle for DL, so Proposition 5.2.4 can be used, and the problem OKD becomes easy.

Remark 5.5.17. For arbitrary key derivation functions, such as those based on SHA-1 as in standardized versions of Menezes–Qu–Vanstone key agreement, it would be very surprising if a key derivation opener could help one to solve the Diffie–Hellman problem. Remark 5.5.16 shows that theoretical helpful openers exist, but it would seem unlikely that SHA-1 could be used to solve discrete logarithms.

So, it seems reasonable to conjecture that, for a typical key derivation function, the openably-derived Diffie–Hellman problem is essentially equivalent to the Diffie–Hellman problem.

If the derived Diffie–Hellman problem is to be hard, then the openably-derived Diffie–Hellman problem must be hard or the key derivation function is seconder-resistant, because of the following proposition.

Proposition 5.5.11. *If* KDH *is a derived Diffie–Hellman problem solver, then algorithm* OKD *defined by*

$$\mathsf{OKD}(P, Q, \mathsf{KDO}) \to \mathsf{KDO}(\mathsf{KDH}(P, Q)) \tag{5.5.20}$$

is an openably-derived Diffie–Hellman problem solver and algorithm KDS *defined by*

$$p \leftarrow \mathbb{Z}_n \tag{5.5.21}$$

$$\mathsf{KDS}(V) \to \mathsf{KDO}(\mathsf{KDH}(pG, p^{-1}V)) \tag{5.5.22}$$

(where p^{-1} *is set to* 1 *if* $p = 0$*) is a key derivation seconder with*

- *success rates such that* $\sigma(\mathsf{OKD}) + \sigma(\mathsf{KDS}) \geqslant (\sigma(\mathsf{KDO})\sigma(\mathsf{KDH}))(1 - \frac{1}{n})$*, and*
- *computational costs* $\gamma(\mathsf{OKD}) = \gamma(\mathsf{KDH})$ *and* $\gamma(\mathsf{KDS}) \leqslant \gamma(\mathsf{KDH}) + 2\log_2(n)\gamma(\$) + 4\log_2(n)\gamma(\mathbb{G})$.

Proof. Algorithm OKD from (5.5.20) above consists of invoking KDH, and calling its oracle KDO (whose cost we do not count in the cost of OKD), so its computational cost is $\gamma(\mathsf{OKD}) = \gamma(\mathsf{KDH})$, as claimed. Algorithm KDS from above consists of invoking algorithm KDH, selecting a random value from $\mathbb{Z}_n$, doing an inversion operation in $\mathbb{Z}_n$ (not included in the cost), and then two scalar multiplications in $\mathbb{G}$, so its computational cost $\gamma(\mathsf{KDS})$ has the claimed upper bound.

To establish the success rates, consider the following process.

$$V \leftarrow \mathbb{G} \tag{5.5.23}$$

$$p \leftarrow \mathbb{Z}_n \tag{5.5.24}$$

$$P \leftarrow pG \tag{5.5.25}$$

$$Q \leftarrow \begin{cases} p^{-1}V & \text{if } p \neq 0 \\ V & \text{if } p = 0 \end{cases} \tag{5.5.26}$$

$$k \leftarrow \mathsf{KDH}(P, Q) \tag{5.5.27}$$

$$R \leftarrow \mathsf{KDO}(k) \tag{5.5.28}$$

$$s_{\mathsf{KDH}} \leftarrow k \mathrel{\hat{=}} d(P \otimes Q) \tag{5.5.29}$$

$$s_{\mathsf{KDO}} \leftarrow d(R) \mathrel{\hat{=}} k \tag{5.5.30}$$

$$s_{\mathsf{KDS}} \leftarrow (d(R) \mathrel{\hat{=}} d(V))(1 - (R \mathrel{\hat{=}} V)) \tag{5.5.31}$$

$$s_{\mathsf{OKD}} \leftarrow R \mathrel{\hat{=}} (P \otimes Q) \tag{5.5.32}$$

First note that P and Q are independently and uniformly distributed in $\mathbb{G}$ in accordance with Definition 5.5.1, which establishes that the expected value of s_{KDH} is $\sigma(\mathsf{KDH})$. Also, P and Q are properly distributed according to Definition 5.5.9, so the expected value of s_{OKD} is $\sigma(\mathsf{OKD})$.

Next consider the event $p \neq 0$, which happens with probability $(1 - \frac{1}{n})$. In this event, then $V = pG \otimes p^{-1}V = P \otimes Q$, and V is also uniformly distributed in $\mathbb{G}$, in accordance with Definition 5.4.4. So, the expected value of s_{KDS} is $\sigma(\mathsf{KDS})$, in this event.

It also follows that in the event $p \neq 0$, the following identity holds:

$$s_{\mathsf{KDS}} = (d(R) \doteq d(P \otimes Q))(1 - (R \doteq (P \otimes Q))) \tag{5.5.33}$$

From this it follows that:

$$\begin{aligned} s_{\mathsf{KDO}} s_{\mathsf{KDH}} &= (d(R) \doteq k)(k \doteq d(P \otimes Q)) \\ &\leqslant (d(R) \doteq d(P \otimes Q)) \\ &= (d(R) \doteq d(P \otimes Q))(R \doteq (P \otimes Q)) + (d(R) \doteq d(P \otimes Q))(1 - (R \doteq (P \otimes Q))) \\ &= (R \doteq (P \otimes Q)) + (d(R) \doteq d(P \otimes Q))(1 - (R \doteq (P \otimes Q))) \\ &= s_{\mathsf{OKD}} + s_{\mathsf{KDS}}. \end{aligned} \tag{5.5.34}$$

When $s_{\mathsf{KDH}} = 1$, then k has distribution required by Definition 5.4.12. So, in the event $p \neq 0$ and $s_{\mathsf{KDH}} = 1$, it follows that the expected value of s_{KDO} is actually $\sigma(\mathsf{KDO})$. The desired bound then follows. □

Proposition 5.5.11 may be summarized as

$$\mathsf{KDH} \Rightarrow (\mathsf{OKD}|\mathsf{KDS}). \tag{5.5.35}$$

which should be contrasted to $(\mathsf{DH}|\mathsf{KDP}) \Rightarrow \mathsf{KDH}$.

Remark 5.5.18. With the components from Remark 5.5.16, the problem OKD becomes easy, yet the corresponding problem KDH appears to remain hard, so there can sometimes be a gap in (5.5.35), with the right side being much easier.

5.5.4 Decisional Derived Diffie–Hellman Problem Hard

Definition 5.5.12. *An algorithm* DKD *is a decisional derived Diffie–Hellman problem solver to the base G, for the group $\mathbb{G}$ and for key derivation function $d : \mathbb{G} \to \mathbb{K}$, if it takes as input two elements in $\mathbb{G}$ and one element of $\mathbb{K}$ and outputs a bit. Its success rate $\sigma(\mathsf{DKD})$ is defined as the expected value of s in the following process:*

$$P, Q \leftarrow \mathbb{G} \tag{5.5.36}$$
$$k \leftarrow \mathbb{K} \vee d(P \otimes Q) \tag{5.5.37}$$
$$b \leftarrow \mathsf{DKD}(P, Q, k) \tag{5.5.38}$$
$$s \leftarrow b \eqcirc (k \doteq d(P \otimes Q)) \tag{5.5.39}$$

The hardness condition associated withe decisional derived Diffie–Hellman problem is that a high-success, low-cost DKD is not constructible.

Remark 5.5.19. This condition is necessary for Menezes–Qu–Vanstone to provide decisional passive implicit key authentication because of Lemma 6.6.2.

For the decisional derived Diffie–Hellman problem to be hard, it must be true that key derivation distinguishing is hard, because of the following proposition.

Proposition 5.5.13. *If* KDD *is a key derivation distinguisher, then algorithm* DKD *defined by*

$$\mathsf{DKD}(P, Q, k) \to \mathsf{KDD}(k) \tag{5.5.40}$$

is a decisional derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{DKD}) \geqslant \sigma(\mathsf{KDD})\left(1 - \frac{1}{n}\right)$, *and*
- *computational cost* $\gamma(\mathsf{DKD}) = \gamma(\mathsf{KDD})$.

Proof. Algorithm DKD from (5.5.40) consists of invoking algorithm KDD, so its computational cost is $\gamma(\mathsf{DKD}) = \gamma(\mathsf{KDD})$, as claimed.

In the process for DKD, Definition 5.5.12, elements P and Q are selected uniformly at random in $\mathbb{G}$. Let $R = P \otimes Q$. The distribution of element R is equivalent to $R \leftarrow 0 \vee_n \mathbb{G}$, where $\vee_n$ is the operator, that like $\vee$, takes two distributions and produces another distribution that is a switch between them. Operator $\vee_n$ chooses the left distribution with probability $\frac{1}{n}$ and the right distribution otherwise.

So, with probability $\frac{n-1}{n}$, element R is distributed in accordance with its distribution for KDD. In this case, KDD has probability almost $\sigma(\mathsf{KDD})$ of resulting in success indicator 1. But the success indicator for KDD and DKD have the same formula, so in this case DKD succeeds too. □

For the decisional derived Diffie–Hellman problem to be hard, it must be true that the derived Diffie–Hellman problem is hard.

Proposition 5.5.14. *If* KDH *is a derived Diffie–Hellman problem solver, then algorithm* DKD *defined by*

$$\mathsf{DKD}(P, Q, k) \to k \stackrel{?}{=} \mathsf{KDH}(P, Q) \tag{5.5.41}$$

is a decisional derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{DKD}) \geqslant \sigma(\mathsf{KDH}) + \frac{2}{|\mathbb{K}|}(1 - \sigma(\mathsf{KDH}))$, *and*
- *computational cost* $\gamma(\mathsf{DKD}) = \gamma(\mathsf{KDH})$.

Proof. Algorithm DKD from (5.5.41) above consists of invoking algorithms KDH, so its computational cost is $\gamma(\mathsf{DKD}) = \gamma(\mathsf{KDH})$, as claimed.

Under the process for DKD in Definition 5.5.12, elements in P and Q are uniformly and independently distributed in $\mathbb{G}$. The success indicator for DKD above may be expressed as

$$(k \stackrel{?}{=} \mathsf{KDH}(P, Q)) \stackrel{?}{\Leftrightarrow} (k \stackrel{?}{=} d(P \otimes Q)) \tag{5.5.42}$$

As inputs P and Q to KDH, they are distributed in accordance with the process for KDH under Definition 5.5.1. So, with probability $\sigma(\mathsf{KDH})$ it holds that $k' = d(P \otimes Q)$, which cause the success indicator of DKD to evaluate to one, because both sides of $\stackrel{?}{\Leftrightarrow}$ in (5.5.42) are the same.

Otherwise, KDH is not successful, which occurs with probability $1 - \sigma(\mathsf{KDH})$, and means $\mathsf{KDH}(P, Q) \neq d(P \otimes Q)$. In this case, the success indicator of DKD is 1 except if $k \in \{\mathsf{KDH}(P, Q), d(P \otimes Q)\}$, in which case it is -1. Because the probability $k \leftarrow \mathbb{K} \vee d(P \otimes Q)$ under the process for DKD, the probability that $k = d(P \otimes Q)$ is $\frac{1}{2} + \frac{1}{2|\mathbb{K}|}$. The probability that $k = \mathsf{KDH}(P, Q)$ is $\frac{1}{2|\mathbb{K}|}$ (because it is now being assumed that KDH is not successful). So,

$$\sigma(\mathsf{DKD}) = \sigma(\mathsf{KDH}) + (1 - \sigma(\mathsf{KDH}))\left(\left(\frac{1}{2} + \frac{1}{|\mathbb{K}|}\right) - \left(\frac{1}{2} - \frac{1}{|\mathbb{K}|}\right)\right), \tag{5.5.43}$$

which simplifies to the claimed success rate. □

Remark 5.5.20. The decisional derived Diffie–Hellman problem is the decisional version of the derived Diffie–Hellman problem.

If the decisional derived Diffie–Hellman problem is to be hard, then the following conjecture suggests that either the decisional Diffie–Hellman problem or key derivation function reversing must be hard.

Conjecture 5.5.15. *If* DDH *is a decisional Diffie–Hellman problem solver and* KDR *is a key derivation reverser, then algorithm* DKD *defined by*

$$\mathsf{DKD}(P, Q, k) \to \mathsf{DDH}(P, Q, \mathsf{KDR}(k)) \tag{5.5.44}$$

is a decisional derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{DKD}) \geqslant \sigma(\mathsf{DDH})\sigma(\mathsf{KDR})$*, and*
- *computational cost* $\gamma(\mathsf{DKD}) = \gamma(\mathsf{DDH}) + \gamma(\mathsf{KDO})$.

The success rate bound in the conjecture above is purely speculative. The formal obstacle that I encountered when trying to find a proof of such a result was supplying the oracle DDH with a input of the correct distribution. The distribution supplied to DKD only partially interfaces with that of KDR input with the ensuing output not seeming to transformable into the correct distribution for DDH. This may be a removable obstacle, an artificial obstacle, or else a essential obstracle reflecting an incorrectness in the conjecture. If I had not gone through the process of writing other proofs in this report and attempting to find a proof for this statement, I probably would have written a proof for this result in the form "Obvious". Even if a proof is "obvious", "obvious" is not a proof.

The following conjecture suggests that if the decisional Diffie–Hellman problem is hard and key derivation distinguishing is hard, then the decisional derived Diffie–Hellman problem is hard.

Conjecture 5.5.16. *If* DKD *is a decisional derived Diffie–Hellman problem solver, then the algorithm* DDH *defined by*

$$\mathsf{DDH}(P, Q, R) \to \mathsf{DKD}(P, Q, d(R)) \tag{5.5.45}$$

is a decisional Diffie–Hellman problem solver, and the algorithm KDD *defined by*

$$P, Q \leftarrow \mathbb{G} \tag{5.5.46}$$

$$\mathsf{KDD}(k) \to \mathsf{DKD}(P, Q, k) \tag{5.5.47}$$

is a key derivation distinguisher, with

- *success rates such that* $\sigma(\mathsf{DDH}) + \sigma(\mathsf{KDD}) \approx \sigma(\mathsf{DKD})$,
- *computational costs* $\gamma(\mathsf{DDH}) = \gamma(\mathsf{DKD}) + \gamma(d)$ *and* $\gamma(\mathsf{KDD}) \leqslant \gamma(\mathsf{DKD}) + 4\log_2(n)(\gamma(\mathbb{G}) + \gamma(\$))$.

Again, a result of this form seemed to very likely to me, but a formal obstacle prevented me from finding a proof.

Imprecise Sketch. As usual, the computational costs are straightforward to verify.

In this sketch, we drop our level of precision regarding success rates down a little, by approximating that $1/n \approx 0$ and $1/|\mathbb{K}| \approx 0$.

Consider the following process:

$$P, Q, U \leftarrow \mathbb{G} \tag{5.5.48}$$
$$R \leftarrow (P \otimes Q) \vee U \tag{5.5.49}$$
$$k \leftarrow \mathbb{K} \vee d(R) \tag{5.5.50}$$
$$b \leftarrow \mathsf{DKD}(P, Q, k) \tag{5.5.51}$$
$$s_{\mathsf{DDH}} \leftarrow b \eqcirc (R \stackrel{\circ}{=} P \otimes Q) \tag{5.5.52}$$
$$s_{\mathsf{KDD}} \leftarrow b \eqcirc (k \stackrel{\circ}{=} d(U)) \tag{5.5.53}$$
$$s_{\mathsf{DKD}} \leftarrow b \eqcirc (k \stackrel{\circ}{=} d(P \otimes Q)) \tag{5.5.54}$$

The variables of the overall process do not have the correct distributions to ensure the expected values of the three success indicates is the correct success rate. But within the following events, the success rates are correct:

- Within the event $k = d(R)$, the expected value of s_{DDH} is $\sigma(\mathsf{DDH})$.
- Within the event $R = U$, the expected value of s_{KDD} is $\sigma(\mathsf{KDD})$.
- Within the event $R = P \otimes Q$, the expected value of s_{DKD} is $\sigma(\mathsf{KDD})$.

The success rates are correct for a similar reason in each of the three cases: the event restricts the variables to the distributions required by the process for that algorithm.

Just to be clear, in the first two events, we are making the substitutions provided (5.5.45) and (5.5.46) into the process for DKD, and then aligning the corresponding parts of the resulting process with the corresponding process for DDH and KDD, respectively.

A fourth event will be useful to consider: the event $k \neq d(R)$. In this event, algorithm DKD is not given any information about R, so it is impossible for DKD is distinguish between the events $R = P \otimes Q$ and $R = U$. So, let c, d, e be the probability that $b = 1$ in the following three respective events:

1. $k = d(R)$ and $R = P \otimes Q$,
2. $k = d(R)$ and $R = U$,
3. $k \neq d(R)$.

Since we assume nothing about DKD, it may be possible that a given DKD could set these values to any probabilities.

Now we proceed to compute the three success rates in terms of the three numbers c, d, e. Again, to simplify the calculations we drop the level of precision, by ignoring some terms of the form $1/n$ and $1/|K|$.

- $\sigma(\mathsf{DDH}) = c - d$, because $\sigma(\mathsf{DDH})$ is the expected value of s_{DDH} in the event that $k = d(R)$, which splits into two events by whether $R = P \otimes Q$ or $R = U$. In the first subevent, if $b = 1$, which has probability c, the value of s_{DDH} is 1. With the probability $1 - c$, the value of s_{DDH} is -1. The expected value of s_{DDH} in this sub-event is $2c - 1$. In the other sub-event, if $b = 1$, then $s_{\mathsf{DDH}} = -1$, which happens with probability d. So, by a similar argument, the expected value of s_{DDH} in this sub-event is $-(2d - 1) = 1 - 2d$. The expected value of s_{DDH} is the average of its expected in the two sub-events.

- $\sigma(\mathsf{KDD}) = d - e$, because $\sigma(\mathsf{KDD})$ is the expected value of s_{KDD} in the event that $R = U$, which can be split into two subevents by whether $d = k(R)$ or $d \neq k(R)$. In the former subevent, we have $s_{\mathsf{KDD}} = b \mathbin{\dot{-}} 1$, becuase $(k \stackrel{\circ}{=} d(U)) = 1$ given the conditions of the event $k = d(R)$ and $R = U$. So, in this subevent, the expect value of s_{KDD} is $2d - 1$. In the other subevent, which is $R = U$ and $d \neq k(R)$, can conclude $k \neq d(U)$, so that the $s_{\mathsf{KDD}} = b \mathbin{\dot{-}} 0$. In this event, the probability that $b = 1$ is e, and the resulting expected value of s_{KDD} is $1 - 2e$. Averaging these two expected values gives $\sigma(s_{\mathsf{KDD}}) = d - e$.
- $\sigma(\mathsf{DKD}) = c - e$, becaue $\sigma(\mathsf{DKD})$ is the expected value of s_{DKD} in the event $R = P \otimes Q$. This event, like the two previous, splits into two subevents based on whether $k = d(R)$ or $k \neq d(R)$. In the former subevent, we have $s_{\mathsf{DKD}} = b \mathbin{\dot{-}} 1$ and the probability that $b = 1$ is c, so the expected value of s_{KDD} is $2c - 1$ in this subevent. In the other subevent, we have $k \neq d(P \otimes Q)$, so $s_{\mathsf{DKD}} = b \mathbin{\dot{-}} 0$. The probability that $b = 1$ in this subevent is e, so the expected value of s_{KDD} is $1 - 2e$. The average over the two subevents is $\sigma(\mathsf{DKD}) = c - e$.

Therefore, $\sigma(\mathsf{DKD}) = \sigma(\mathsf{DDH}) + \sigma(\mathsf{KDD})$, as desired. □

If a precise proof of this conjecutre is possible, then it seems plausible that its success rates would be off by a difference of small amounts such as $\frac{1}{|\mathbb{K}|}$ and $\frac{1}{n}$.

Admittedly, this proof sketch seems a little too complicated for such a simple idea. In particular, the proof is peculiarly artificial and indirect in considering unknown probabilities of algorithms playing games that they were not designed to play.

Presumably, a similar result has already been proven in work previous to this report. Furthermore, the proof of such result is likely to be simpler and more direct than this report's. Indeed, it seems that such a result ought to have been well-known from the dawn of public-key cryptography, or at least provable security. If so, then alas, the author of this report is not well-informed.

I was motivated to provide the proof sketch above by Watson Ladd's presentation to the Cryptographers' Forum Research Group in which a result similar to the conjecture above was claimed.

5.6 Joint Conditions on the Group and Conversion Function

5.6.1 One-Up Problem Hard

Definition 5.6.1. *A one-up problem solver in $\mathbb{G}$ with conversion function $f : \mathbb{G} \to \mathbb{Z}_n$ is an algorithm* OUP *whose input is two elements of $\mathbb{G}$ and whose output is one element of $\mathbb{G}$. Its success rate $\sigma(\mathsf{OUP})$ is defined as the expected value of s in the following process:*

$$P, Q \leftarrow \mathbb{G} \tag{5.6.1}$$
$$R \leftarrow \mathsf{OUP}(P, Q) \tag{5.6.2}$$
$$s \leftarrow R \stackrel{\circ}{=} P + f(R)Q \tag{5.6.3}$$

The hardness condition associated with a one-up problem solver is that a high-success low-cost OUP is not constructible.

Remark 5.6.1. The one-up problem was introduced in [Bro08] in the context of the elliptic curve digital signature algorithm (ECDSA), which, like MQV, uses a conversion function $f : \mathbb{G} \to \mathbb{Z}_n$, but uses a different conversion function.

The conversion function must be non-predicting if it is to be true that the one-up problem is hard, because of the following proposition.

Proposition 5.6.2. *Let* CFP *be conversion function predictor for conversion function* $f : \mathbb{G} \to \mathbb{Z}_n$*, then the algorithm* OUP *defined by*

$$\mathsf{OUP}(P, Q) \to P + \mathsf{CFP}()Q \tag{5.6.4}$$

is a one-up problem solver for group $\mathbb{G}$ *and conversion function* $f : \mathbb{G} \to \mathbb{Z}_n$ *with*

- *success rate* $\sigma(\mathsf{OUP}) \geqslant \sigma(\mathsf{CFP})$*, and*
- *computational cost* $\gamma(\mathsf{OUP}) \leqslant \gamma(\mathsf{CFP}) + (1 + 2\log_2(n))\gamma(\mathbb{G})$.

Proof. Algorithm OUP above invokes CFP, and does one scalar multiplication and one addition in $\mathbb{G}$, so its cost $\gamma(\mathsf{OUP})$ is as stated above.

In the process for OUP (Definition 5.6.1) elements P and Q are uniformly distributed in $\mathbb{G}$. Let $r \leftarrow \mathsf{CFP}()$ in execution of OUP above. Because CFP() does not depend on P or Q, element $R = P + rQ$ is uniformly distributed in $\mathbb{G}$. In the general process for CFP (Definition 5.3.1, the probability that $r = f(R)$ is $\sigma(\mathsf{CFP})$. In this case, OUP has success indicator $s = 1$. Therefore, $\sigma(\mathsf{OUP}) \geqslant \sigma(\mathsf{CFP})$. □

Remark 5.6.2. By taking a (possibly low-success) CFP that outputs an element of $\mathbb{Z}_n$ uniformly at random, repeating the algorithm OUP from Proposition 5.6.2 approximately $|f(\mathbb{G})|$ times, one gets a fairly high-success rate one-up problem solver of cost proportional approximately $|f(\mathbb{G})|(2\log_2(n))\gamma(\mathbb{G})$.

Remark 5.6.3. It is conjectured in [Bro08] that, provided f meets certain conditions, that the ECDSA version of the one-up problem takes n group operations to solve. Based on the Remark 5.6.2, a more general conjecture would be that any high-success OUP has cost at least approximately $|f(\mathbb{G})|\gamma(\mathbb{G})$.

Remark 5.6.4. It is noted in [Bro08] that, over general groups, the one-up problem and the discrete logarithm are independent problems, in that one could be hard while the other could be easy.

Remark 5.6.5. If it remains true that no known algorithm converts DL into a OUP, then any result that MQV resists CAIKA must assume that both DL and OUP are hard problems.

5.6.2 Semi-Logarithm Problem Hard

Definition 5.6.3. *A semi-logarithm solver for* $\mathbb{G}$ *with conversion function* f *to the base* G *is an algorithm* SL *with input of one element of* $\mathbb{G}$ *and output of two integers. Its success rate* $\sigma(\mathsf{SL})$ *is defined to be the expected value of* s *in the process:*

$$P \leftarrow \mathbb{G} \tag{5.6.5}$$

$$(t, u) \leftarrow \mathsf{SL}(P) \tag{5.6.6}$$

$$s \leftarrow t \stackrel{?}{=} f(uG - tP) \tag{5.6.7}$$

The hardness condition associated with semi-logarithm solvers is that a high-success, low-cost SL is not constructible.

Remark 5.6.6. Semi-logarithms were first defined for ECDSA. They were defined similarly, except that the defining equation was $t = f((u^{-1} \bmod n)(G + tP))$, or, in other words, $t = f(vG + tvP)$ where $uv \equiv 1 \bmod n$. The hardness of computing semi-logarithms was shown to be necessary for the security of ECDSA.

Both the discrete logarithm problem and the one-up problem must be hard if it is to be true that the semi-logarithm problem is hard, because of the following two propositions.

Proposition 5.6.4. *If* DL *is a discrete logarithm solver in the group* $\mathbb{G}$ *to the base* G*, then the algorithm* SL *defined by*

$$z \leftarrow \mathbb{Z}_n \tag{5.6.8}$$
$$t \leftarrow f(zG) \tag{5.6.9}$$
$$\mathsf{SL}(P) \rightarrow (t, z + t\mathsf{DL}(P)) \tag{5.6.10}$$

is a semi-logarithm solver for $\mathbb{G}$ *to the base* G *and conversion function* $f : \mathbb{G} \rightarrow \mathbb{Z}_n$ *with*

- *success rate* $\sigma(\mathsf{SL}) \geqslant \sigma(\mathsf{DL})$*, and*
- *computational cost* $\gamma(\mathsf{SL}) \leqslant \gamma(\mathsf{DL}) + \gamma(f) + (2\log_2(n))\gamma(\mathbb{G}) + 2\log_2(n)\gamma(\$)$.

Proof. Algorithm SL above invokes algorithm DL, evaluates function f, selects a random element of $\mathbb{Z}_n$, and does one scalar multiplication in $\mathbb{G}$, is its computational cost $\gamma(\mathsf{SL})$ has the upper bound claimed above.

The success indicator for this algorithm SL, according Definition 5.6.3, is

$$t \stackrel{\circ}{=} f((z + t\mathsf{DL}(P))G - tP) \tag{5.6.11}$$

The input P to DL, provided by SL, is uniformly distributed in $\mathbb{G}$, in accordance with the Definition 5.2.1, because under the Definition 5.6.3, P is uniformly distributed in $\mathbb{G}$. So, with probability $\sigma(\mathsf{DL})$, it holds that $P = \mathsf{DL}(P)G$. In this event, the success indicator for SL, as given by (5.6.11), evaluates to 1, because the left side of the operator $\stackrel{\circ}{=}$ is $t = f(zG)$ and the right side is $f(zG + t\mathsf{DL}(P)G - t\mathsf{DL}(P)G) = f(zG)$, which are the same. □

Proposition 5.6.5. *If* OUP *is a one-up problem solver for the group* $\mathbb{G}$ *and conversion function* $f : \mathbb{G} \rightarrow \mathbb{Z}_n$*, then the algorithm* SL *defined by*

$$u \leftarrow \mathbb{Z}_n \tag{5.6.12}$$
$$R \leftarrow \mathsf{OUP}(uG, -P) \tag{5.6.13}$$
$$\mathsf{SL}(P) \rightarrow (f(R), u) \tag{5.6.14}$$

is a semi-logarithm solver for $\mathbb{G}$ *to the base* G *and conversion function* $f : \mathbb{G} \rightarrow \mathbb{Z}_n$ *with*

- *success rate* $\sigma(\mathsf{SL}) \geqslant \sigma(\mathsf{OUP})$*, and*
- *computational cost* $\gamma(\mathsf{SL}) \leqslant \gamma(\mathsf{OUP}) + \gamma(f) + (2\log_2(n))\gamma(\mathbb{G}) + 2\log_2(n)\gamma(\$)$.

Proof. Algorithm SL above invokes algorithm OUP, evaluates function f, selects a random element of $\mathbb{Z}_n$, and does a scalar multiplication in the group $\mathbb{G}$, so its computational cost $\gamma(\mathsf{SL})$ has upper bound as claimed above.

The success indicator for the algorithm SL above, according to Definition 5.6.3, is given by

$$f(R) \stackrel{\circ}{=} f(uG - f(R)P) \tag{5.6.15}$$

The inputs uG and $-P$ provided by SL to OUP are independently and uniformly distributed in $\mathbb{G}$, in accordance with the process for OUP (Definition 5.6.1), because of the way that SL selects uG and because P provided to SL under Definition 5.6.3 uniformly at random from $\mathbb{G}$. So, with probability $\sigma(\mathsf{OUP})$, it is true that $R = uG - f(R)P$. In this case, applying f to both sides of this equation, shows that both sides of the operator $\mathrel{\mathring{=}}$ in the success indicator for SL, as given by (5.6.15), are equal. So $\sigma(\mathsf{SL}) \geqslant \sigma(\mathsf{OUP})$. □

To summarize these last two relations, we can write

$$(\mathsf{DL}|\mathsf{OUP}) \Rightarrow \mathsf{SL} \tag{5.6.16}$$

Remark 5.6.7. The MQV semi-logarithm problem is potentially easier than both discrete logarithm problem and the one-up problem.

Remark 5.6.8. The MQV semi-logarithm problem with t fixed is closely related to the discrete logarithm. The MQV semi-logarithm problem with u fixed is closely related to the one-up problem.

5.6.3 Converted Diffie–Hellman Problem Hard

Definition 5.6.6. *An algorithm* CDH *is a converted Diffie–Hellman problem solver to the base G for the group $\mathbb{G}$ if it takes as input two elements of $\mathbb{G}$ and outputs two elements of $\mathbb{G}$. Its success rate $\sigma(\mathsf{CDH})$ is defined as the expected value of s in the following process:*

$$P, Q \leftarrow \mathbb{G} \tag{5.6.17}$$
$$(Y, U) \leftarrow \mathsf{CDH}(P, Q) \tag{5.6.18}$$
$$s \leftarrow U \mathrel{\mathring{=}} (P \otimes (Y + f(Y)Q)) \tag{5.6.19}$$

For the converted Diffie–Hellman problem to be hard it must be true that both the Diffie–Hellman problem and the MQV semi-logarithm problem are hard, because of the following two propositions.

Proposition 5.6.7. *If* DH *is a Diffie–Hellman problem solver, then algorithm* CDH *defined by*

$$Y \leftarrow \mathbb{G} \tag{5.6.20}$$
$$\mathsf{CDH}(P, Q) \rightarrow (Y, \mathsf{DH}(P, Y + f(Y)Q)) \tag{5.6.21}$$

is a converted Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{CDH}) \geqslant \sigma(\mathsf{DH})$, *and*
- *computational cost* $\gamma(\mathsf{CDH}) \leqslant \gamma(\mathsf{DH}) + \gamma(f) + (2\log_2(n))\gamma(\$) + (1 + 2\log_2(n)\gamma(\mathbb{G})$.

Proof. Algorithm CDH invokes algorithm DH, evaluates function f, selects a random element of $\mathbb{Z}_n$, and does one scalar multiplication and one addition in $\mathbb{G}$, so $\gamma(\mathsf{CDH})$ has the claimed upper bound.

The success indicator for algorithm CDH above, according to Definition 5.6.6, is

$$\mathsf{DH}(P, Y + f(Y)Q) \mathrel{\mathring{=}} (P \otimes (Y + f(Y)Q)) \tag{5.6.22}$$

The inputs P and $Y + f(Y)Q$ that CDH provides to DH are independently and uniformly distributed in $\mathbb{G}$, in accordance with the process for DH (Definition 5.2.3), because under the process for CDH, elements P and Q are independently and uniformly distributed, and under the algorithm CDH, element Y is independent of Q. So, with probability $\sigma(\mathsf{DH})$, the output of DH is $P \otimes (Y + f(Y)Q)$. In this event, the success indicator for CDH, as given by (5.6.22) is 1. Therefore, $\sigma(\mathsf{CDH}) \geqslant \sigma(\mathsf{DH})$. □

The semi-logarithm problem must be hard if the converted Diffie–Hellman problem is to be hard, because of the following proposition.

Proposition 5.6.8. *If* SL *is a semi-logarithm solver, then algorithm* CDH *defined by*

$$(t, u) \leftarrow \mathsf{SL}(Q) \tag{5.6.23}$$
$$\mathsf{CDH}(P, Q) \rightarrow (uG - tQ, uP) \tag{5.6.24}$$

is a converted Diffie–Hellman solver with

- *success rate* $\sigma(\mathsf{CDH}) \geqslant \sigma(\mathsf{SL})$, *and*
- *computational cost* $\gamma(\mathsf{CDH}) \leqslant \gamma(\mathsf{SL}) + (1 + 6\log_2(n))\gamma(\mathbb{G})$.

Proof. Algorithm CDH above invokes algorithm SL, and does one group operation and three scalar multiplications $\mathbb{G}$, so its computational cost $\gamma(\mathsf{CDH})$ has the claimed upper bound.

The input Q is uniformly distributed in $\mathbb{G}$, in accordance with process for SL (Definition 5.6.3) because Q is uniformly distributed in $\mathbb{G}$ under the process for CDH (Definition 5.6.6). So, with probability, $\sigma(\mathsf{SL})$ the success indicator for SL is 1, which means that $t = f(uG - tQ)$.

The success indicator for the algorithm CDH above, under the general process for CDH, can be expressed as

$$uP \stackrel{\circ}{=} (P \otimes (uG - tQ + f(uG - tQ)Q)). \tag{5.6.25}$$

When SL is successful the right side of (5.6.25) simplifies to uP, so CDH succeeds too. Therefore $\sigma(\mathsf{CDH}) \geqslant \sigma(\mathsf{SL})$. □

5.6.4 Distinguish-Up Problem Hard

Definition 5.6.9. *A distinguish-up problem solver in* $\mathbb{G}$ *with conversion function* $f : \mathbb{G} \rightarrow \mathbb{Z}_n$ *is a pair* $\mathsf{DUP} = (\mathsf{DUP}_1, \mathsf{DUP}_2)$ *of algorithm where:* DUP_1 *takes input of one element of* $\mathbb{G}$ *and produces output of one element of* $\mathbb{G}$*; and* DUP_2 *takes input of one element in* $\mathbb{G}$ *and outputs a single bit. Its success rate* $\sigma(\mathsf{DUP})$ *is defined as the expected value of* s *in the following process:*

$$B \leftarrow \mathbb{G} \tag{5.6.26}$$
$$Y \leftarrow \mathsf{DUP}_1(B) \tag{5.6.27}$$
$$Z \leftarrow \mathbb{G} \vee (Y + f(Y)B) \tag{5.6.28}$$
$$b \leftarrow \mathsf{DUP}_2(Z) \tag{5.6.29}$$
$$s \leftarrow b \eqcirc (Z \stackrel{\circ}{=} Y + f(Y)B) \tag{5.6.30}$$

The hardness condition associated with a distinguish-up problem solver is that a high-success low-cost DUP is not constructible.

5.7 Complex Conditions

5.7.1 Reversibly-Derived Converted Diffie–Hellman Problem Hard

Definition 5.7.1. *An algorithm* RCD *is a reversibly-derived converted Diffie–Hellman problem solver if it takes as input two elements of* $\mathbb{G}$ *and an oracle key derivation reverser, and outputs two elements of* $\mathbb{G}$*. Its*

success rate is defined by the expected value of s in the following process:

$$P, Q \leftarrow \mathbb{G} \tag{5.7.1}$$
$$(Y, U) \leftarrow \mathsf{RCD}(P, Q, \mathsf{KDR}) \tag{5.7.2}$$
$$s \leftarrow U \stackrel{?}{=} (P \otimes (Y + f(Y)Q)) \tag{5.7.3}$$

The hardness condition associated with a reversibly-derived converted Diffie–Hellman problem solver is a high-success, low-cost RCD is not constructible. The reversibly-derived converted Diffie–Hellman problem may be summarized as:

$$\mathsf{RCD} = (\mathsf{KDR} \Rightarrow \mathsf{CDH}) \tag{5.7.4}$$

Remark 5.7.1. As in the case of the reversibly-derived Diffie–Hellman problem, it would seem reasonable to assume that, for an arbitrary key derivation function not related to the group $\mathbb{G}$ or the conversion function f, that the KDR oracle is not helpful for solving the converted Diffie–Hellman problem.

5.7.2 Openably-Derived Converted Diffie–Hellman Problem Hard

Definition 5.7.2. *An algorithm OCD is an openably-derived converted Diffie–Hellman problem solver if it takes as input two elements of $\mathbb{G}$ and an oracle key derivation opener, and outputs two elements of $\mathbb{G}$. Its success rate is defined by the expected value of s in the following process:*

$$P, Q \leftarrow \mathbb{G} \tag{5.7.5}$$
$$(Y, U) \leftarrow \mathsf{OCD}(P, Q, \mathsf{KDO}) \tag{5.7.6}$$
$$s \leftarrow U \stackrel{?}{=} (P \otimes (Y + f(Y)Q)) \tag{5.7.7}$$

The hardness condition associated with an openably-derived converted Diffie–Hellman problem solver is a high-success, low-cost OCD is not constructible. The openably-derived converted Diffie–Hellman problem may be summarized as:

$$\mathsf{OCD} = (\mathsf{KDO} \Rightarrow \mathsf{CDH}) \tag{5.7.8}$$

Remark 5.7.2. As in the case of the openably-derived Diffie–Hellman problem, it would seem reasonable to assume that, for an arbitrary key derivation function not related to the group $\mathbb{G}$ or the conversion function d, that the KDO oracle is not helpful for solving the converted Diffie–Hellman problem.

6 Necessity of Certain Conditions on MQV Components for IKA

Necessary isolated conditions on the group are considered in §6.2; necessary isolated conditions on the conversion function are considered in §6.3; and necessary isolated conditions on the key derivation function are considered in §6.4.

Remark 6.0.3. Proving that a condition is necessary for security generally involves demonstrating a reduction between two problems. This reduction is in the opposite direction from the usual direction in provable security, and perhaps could be expressed by the term *conditional security*. It is natural to ask what the benefit of such a conditional security result is.

Occasionally, a conditional security result may uncover a new problem about some of components of the cryptographic system, say hash functions, that has not been considered before. Also, if the effort spent on conditional security was sufficiently substantial, then some assurance may be provided that further effort will not give rise to attacks, or at least attacks based on the components being considered.

Necessary joint conditions on the group and conversion function are considered in §6.5. Necessary joint conditions on the group and key derivation function are considered §6.6. No inseparable joint conditions on the conversion function and the key derivation function related to the security of MQV have been considered in this report.

Remark 6.0.4. If each isolated component condition of a necessary separable joint condition is necessary on its own for a security property, then the conjunction of the isolated conditions is necessary for the conjunction of the security conditions. (We may not state all such logical combination results explicitly). Some separable conditions may be more necessary than the logical combination results would imply. Some joint conditions seem inseparable, in that they do not appear to be expressed as a logical combination of isolated conditions.

Remark 6.0.5. If the component conditions of separable complex conditions are necessary conditions for certain security properties, then the complex condition is necessary for the corresponding logical combination of the security properties. We may not explicitly state all such necessary separable complex conditions. Another class of necessary complex conditions are the security properties themselves. No necessary inseparable complex conditions, other than the security conditions themselves, have been identified in this work.

Remark 6.0.6. Each result in this section is immediate[19] in the sense of being an algorithm contributing computational cost roughly that of the Menezes–Qu–Vanstone key agreement itself to achieve a reduction of some grade of implicit key authentication adversary to the some other problem.

6.1 Lesser Forms of Necessity

A partially-necessary condition is such that it or another condition are necessary. In other words, a partially-necessary condition is a constituent condition of the necessary compound condition, where the compound is a logical disjunction.

A conditionally-necessary condition is such that if the condition fails to hold, then either some grade of implicit key authentication attack on Menezes–Qu–Vanstone key agreement is possible, or another problem is efficiently solvable (typically a problem involving one of the components). A conditionally-necessary condition may not be necessary. If the other problem is believed to be hard, then the conditional-necessity of the conditionally-necessary condition may be considered reasonably close to being a necessary condition.

[19] An example of non-immediate result could be the Leadbitter–Smart attack, which given some oracles, including an oracle for certain bits of the ephemeral private keys, computes a static private key. Non-immediacy refers to the Leadbitter–Smart attack having substantially greater cost than the MQV key agreement.

A sufficiently-necessary condition is such that, if the condition holds, then some other necessary condition also holds. If a sufficiently-necessary condition, such as collision resistance of a hash function, has already been studied thoroughly and accepted as likely, perhaps in other contexts, then the sufficiently-necessary may provide some form of assurance for a condition necessary for the security of Menezes–Qu–Vanstone key agreement.

6.2 Necessary Group-Isolated Conditions

Two isolated conditions on the group component of Menezes–Qu–Vanstone are identified to be necessary for MQV to provide computational passive implicit key authentication. The necessity of these conditions is well-known, especially the first that discrete logarithms are hard for the group.

6.2.1 Discrete Logarithms Hard

The discrete logarithm problem, §5.2.1 must be hard, or else Menezes–Qu–Vanstone key agreement would fall to a computational passive implicit key authentication attack, such as by Eve solving the DLP on Alice's both public keys to obtain both her private keys, or as in the following lemma.

Lemma 6.2.1. *If* DL *is a discrete logarithm solver, then the algorithm* CPIKA *defined by*

$$\mathsf{CPIKA}(A, B, X, Y) \to d(\mathsf{DL}(X + f(X)A)(Y + f(Y)B)) \tag{6.2.1}$$

is a computational passive implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{CPIKA}) \geqslant \sigma(\mathsf{DL})$, *and*
- *computational cost* $\gamma(\mathsf{CPIKA}) \leqslant \gamma(\mathsf{DL}) + (2 + 6\log_2(n))\gamma(\mathbb{G}) + 2\gamma(f) + \gamma(d)$.

Proof. The three scalar multiplications in (6.2.1) each have cost at most $2\log_2(n)\mathbb{G}$ because of double-and-add algorithms.

Let $q = \mathsf{DL}(X + f(X)A)$. With probability $\sigma(DL)$ we have the $X + f(X)A = qG$. In this case, we have $\mathsf{CPIKA}(A, B, X, Y) = d(q(Y + f(Y)B) = \mathsf{mqv}(A, B, X, Y)$, so $s = 1$. Therefore, $\sigma(\mathsf{CPIKA}) \geqslant \sigma(\mathsf{DL})$. □

Remark 6.2.1. An indirect argument for the necessity of a hard discrete logarithm problem is to combine Proposition 5.2.4 that $\mathsf{DL} \Rightarrow \mathsf{DH}$ with Lemma 6.2.2 that $\mathsf{DH} \Rightarrow \mathsf{CPIKA}$.

6.2.2 Diffie–Hellman Problem Hard

The Diffie–Hellman problem must be hard, or else Menezes–Qu–Vanstone key agreement would fall to a computational passive implicit key authentication attack, as in the following lemma.

Lemma 6.2.2. *If* DH *is a Diffie–Hellman problem solver, then the algorithm* CPIKA *defined by*

$$\mathsf{CPIKA}(A, B, X, Y) \to d(\mathsf{DH}(X + f(X)A, Y + f(Y)B)) \tag{6.2.2}$$

is a computational passive implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{CPIKA}) \geqslant \sigma(\mathsf{DH})$ *and,*
- *computational cost* $\gamma(\mathsf{CPIKA}) \leqslant \gamma(\mathsf{DH}) + (2 + 4\log_2(n))\gamma(\mathbb{G}) + 2\gamma(f) + \gamma(d)$.

Proof. The two scalar multiplications in (6.2.2) have cost at most $2\log_2(n)\gamma(\mathbb{G})$. The other operations in CPIKA are an evaluation of DH, two evaluations of f, and one evaluation of d. Therefore, $\gamma(\mathsf{CPIKA})$ is bounded as stated.

Let $R = \mathsf{DH}(X + f(X)A, Y + f(Y)B)$. With probability $\sigma(DH)$ we have the $R = (X + f(X)A) \otimes (Y + f(Y)B)$, because $X + f(X)$ and $Y + f(Y)B$ are each uniformly distributed in $\mathbb{G}$. In this event, $\mathsf{CPIKA}(A, B, X, Y) = \mathsf{mqv}(A, B, X, Y)$, so $s = 1$. Therefore, $\sigma(\mathsf{CPIKA}) \geqslant \sigma(\mathsf{DH})$. □

Remark 6.2.2. Lemma 6.2.2 essentially subsumes the result of Lemma 6.2.1, because of Proposition 5.2.4.

Remark 6.2.3. The success rate of CPIKA could exceed that of DH. For example, if d is a constant function, then CPIKA in (6.2.2) would have success rate 1 regardless of the success rate of DH.

6.3 Necessary Conversion-Function-Isolated Conditions

One isolated condition on the conversion function is identified to be necessary for MQV to provide computational active implicit key authentication.

6.3.1 Conversion Function Predicting Hard

If the conversion function f is predictable (in the sense that one can construct a successful CFP from Definition 5.3.1), then MQV would fall to a computational active IKA attack, as in the following lemma.

Lemma 6.3.1. *If* CFP *is a conversion function predictor, then the computational active implicit key authentication adversary* CAIKA *defined by the process*

$$z \leftarrow \mathbb{Z}_n \tag{6.3.1}$$

$$\mathsf{CAIKA}(A, B, X) \to (zG - \mathsf{CFP}()B, d(z(X + f(X)A)), \tag{6.3.2}$$

has

- *success rate* $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{CFP})$, *and*
- *computational cost* $\gamma(\mathsf{CAIKA}) \leqslant \gamma(\mathsf{CFP}) + \gamma(f) + \gamma(d) + +(2 + 8\log_2(n))\gamma(\mathbb{G}) + 2\log_2(n)\gamma(\$)$.

Proof. Algorithm CAIKA above invokes CFP, evaluates function f and d, does four scalar multiplications and two additions in $\mathbb{G}$, and selects a random number in $\mathbb{Z}_n$, therefore the cost $\gamma(\mathsf{CAIKA})$ is as stated above.

Let j be output of CFP() in the algorithm CAIKA above. Let $Y = zG - jB$, which is uniformly distributed in $\mathbb{G}$ because z is uniformly distributed in $\mathbb{Z}_n$. Therefore, with probability $\sigma(\mathsf{CFP})$, it is true that $f(Y) = j$. Let $k = d(z(X + f(X)A)$, so that the output of CAIKA is (Y, k). Then

$$\begin{aligned} k &= d(z(X + f(X)A)) \\ &= d(zG \otimes (X + f(X)A)) \\ &= d((X + f(X)A) \otimes zG) \\ &= d((x + f(X)a)(Y + jB)) \\ &= d((x + f(X)a)(Y + f(Y)B)) \\ &= \mathsf{mqv}(A, B, X, Y), \end{aligned} \tag{6.3.3}$$

so $s = 1$ in the process (Definition 4.2.1) for CAIKA with probability at least $\sigma(\mathsf{CFP})$. Therefore $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{CFP})$. □

6.4 Necessary Key-Derivation-Function-Isolated Conditions

6.4.1 Key Derivation Function Predicting Hard

If the key derivation d is predicting (in the sense a successful KDP from Definition 5.4.1), then MQV would fall to a computational passive IKA attack, as in the following lemma.

Lemma 6.4.1. *If* KDP *is a key derivation function predictor, then the computational passive implicit key authentication adversary* CPIKA *defined by the process*

$$\mathsf{CPIKA}(A, B, X, Y) \to \mathsf{KDP}(), \tag{6.4.1}$$

has

- *success rate* $\sigma(\mathsf{CPIKA}) \geqslant \sigma(\mathsf{KDP})$, *and*
- *computational cost* $\gamma(\mathsf{CPIKA}) = \gamma(\mathsf{KDP})$.

Proof. Algorithm CPIKA consists of executing KDP, so $\gamma(\mathsf{CPIKA}) = \gamma(\mathsf{KDP})$.

Let $U = (X + f(X)A) \otimes (Y + f(Y)B)$ in the process for CPIKA (Definition 4.1.1). Then s in the process for CPIKA is computed as $k \stackrel{?}{=} d(U)$. The value U is uniformly distributed in $\mathbb{G}$ because B is. According to the process for KDP (Definition 5.4.1), the expected value of s is $\sigma(\mathsf{KDP})$. So $\sigma(\mathsf{CPIKA}) = \sigma(\mathsf{KDP})$. □

Because of Proposition 5.4.8 and Lemma 6.4.1, the collision-resistance of the key derivation function is a sufficiently-necessary condition. This report has not identified it to be a necessary condition.

6.4.2 Key Derivation Function Distinguishing Hard

If key derivation function d is distinguishable (in the sense a successful key derivation distinguisher KDD from Definition 5.4.2 is constructible), then MQV would fall to a decisional passive IKA attack, as in the following lemma.

Lemma 6.4.2. *If* KDD *is a key derivation function distinguisher, then the decisional passive implicit key authentication adversary* DPIKA *defined by the process*

$$\mathsf{DPIKA}(A, B, X, Y, k) \to \mathsf{KDD}(k), \tag{6.4.2}$$

has

- *success rate* $\sigma(\mathsf{DPIKA}) \geqslant \sigma(\mathsf{KDD})(1 - \frac{1}{n})$, *and*
- *computational cost* $\gamma(\mathsf{DPIKA}) = \gamma(\mathsf{KDD})$.

Proof. Algorithm DPIKA only invokes KDD, so therefore its $\gamma(\mathsf{DPIKA}) = \gamma(\mathsf{KDD})$.

In the general process for DPIKA (Definition 4.3.1, let $U = (X + f(X)A) \otimes (Y + f(Y)B)$, which means that $\mathsf{mqv}(A, B, X, Y) = d(U)$. In the event $X + f(X)A \neq 0$, which happens with probability $1/n$, the distribution of U is uniform in $\mathbb{G}$. The success indicator s in the general process for DPIKA is therefore $b \stackrel{?}{=} (k \stackrel{?}{=} d(U))$, in this even, and with this notation for U.

Algorithm DPIKA in (6.4.2) computes $b = \mathsf{KDD}(k)$. The general process for KDD (Definition 5.4.2) implies that its success indicator s, which is defined to be $b \stackrel{?}{=} (k \stackrel{?}{=} d(U))$ has expected value $\sigma(\mathsf{KDD})$.

Because the success indicators for DPIKA and KDD have the same formula with the same distribution of inputs, they have the same expected result. Therefore $\sigma(\mathsf{DPIKA}) = \sigma(\mathsf{KDD})$. □

6.5 Necessary Joint Conditions on the Group and Conversion Function

In this section, inseparable joint conditions on combination of group and conversion function are shown to be necessary for Menezes–Qu–Vanstone key agreement to provide computational active implicit key authentication.

6.5.1 One-Up Problem Hard

The one-up problem (Definition 5.6.1) must be hard, or else MQV would fall to the following a computational active IKA attack because of the following lemma.

Lemma 6.5.1. *If* OUP *is an one-up problem solver for group* $\mathbb{G}$ *with conversion function* $f : \mathbb{G} \to \mathbb{Z}_n$*, then the algorithm* CAIKA *defined by*

$$z \leftarrow \mathbb{Z}_n \tag{6.5.1}$$

$$\mathsf{CAIKA}(A, B, X) \to (\mathsf{OUP}(zG, -B), d(z(X + f(X)A)) \tag{6.5.2}$$

is a computational active implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{OUP})$*, and*
- *computational cost* $\gamma(\mathsf{CAIKA}) \leqslant \gamma(\mathsf{OUP}) + \gamma(d) + \gamma(f) + +(2 + 6\log_2(n))\gamma(\mathbb{G}) + (2\log_2(n))\gamma(\$)$.

Proof. Algorithm CAIKA above invokes OUP, evaluates d and f, computes three scalar multiplication, one addition and one negation in $\mathbb{G}$, and selects one element in $\mathbb{Z}_n$, so the cost $\gamma(\mathsf{CAIKA})$ is as stated above.

The inputs to OUP as used in CAIKA above are zG and $-B$ which are uniformly distributed in $\mathbb{G}$ because z is uniformly distributed in $\mathbb{Z}_n$ and B is uniformly distributed in $\mathbb{G}$ by the definition of the general process for CAIKA (Definition 4.2.1). Let $Y \leftarrow \mathsf{OUP}(zG, -B)$. By the general process for OUP (Definition 5.6.1), the probability that $Y = zG - f(Y)B$ is $\sigma(\mathsf{OUP})$. In this event, $zG = Y + f(Y)B$.

So, with probability $\sigma(\mathsf{OUP})$, the output of CAIKA above is (Y, k) such that $k = d(z(X + f(X)A)) = d((Y + f(Y)) \otimes (X + f(X)A)) = \mathsf{mqv}(A, B, X, Y)$. Therefore $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{OUP})$. □

Remark 6.5.1. If it remains true that no known algorithm converts DL into a OUP, then any result that MQV resists CAIKA must assume that both DL and OUP are hard problems.

6.5.2 Semi-logarithms Hard

The MQV semi-logarithm problem (Definition 5.6.3) must be hard, or else MQV could fall to a computational active IKA attack (Definition 4.2.1), as in the following lemma.

Lemma 6.5.2. *If* SL *is an MQV semi-logarithm solver, then the computational active implicit key authentication adversary* CAIKA *defined by*

$$(t, u) \leftarrow \mathsf{SL}(B) \tag{6.5.3}$$

$$\mathsf{CAIKA}(A, B, X) \to (uG - tB, d(u(X + f(X)A))) \tag{6.5.4}$$

has

- *success rate* $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{SL})$*, and*
- *computational cost* $\sigma(\mathsf{CAIKA}) \leqslant \sigma(\mathsf{SL}) + \gamma(d) + \gamma(f) + (2 + 8\log_2(n))\gamma(\mathbb{G})$.

Proof. Algorithm CAIKA invokes SL, evaluates d and f, and does four scalar multiplications and two additions in the group $\mathbb{G}$, so its cost $\gamma(\mathsf{CAIKA})$ is as stated above.

The general process for CAIKA (Definition 4.2.1), and applying to the algorithm CAIKA above, we have $Y = uG - tB$ and $k = d(u(X + f(X)A))$.

Because B is uniformly distributed in $\mathbb{G}$ due to the process for CAIKA, the input to SL is uniformly distributed in accordance to the process for SL (Definition 5.6.3). Consequently, $t = f(uG - tB)$ with probability $\sigma(\mathsf{SL})$.

Because $Y = uG - tP$, it follows that $uG = Y + tB$. As noted above, with probability $\sigma(\mathsf{SL})$, we have $t = f(uG - tB) = f(Y)$ implying that $uG = Y + f(Y)B$ with probability $\sigma(\mathsf{SL})$.

So, with probability $\sigma(\mathsf{SL})$,

$$\begin{aligned} k &= d(u(X + f(X)A)) \\ &= d(uG \otimes (X + f(X)A)) \\ &= d((Y + f(Y)B) \otimes (X + f(X)A)) \\ &= \mathsf{mqv}(A, B, X, Y) \end{aligned} \tag{6.5.5}$$

and CAIKA succeeds with $s = 1$. Therefore, $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{SL})$. □

6.5.3 Converted Diffie–Hellman Problem Hard

The converted Diffie–Hellman problem (Definition 5.6.6) must be hard, or else MQV would fall to a computational active implicit key authentication attack (Definition 4.2.1), as in the following lemma.

Lemma 6.5.3. *If* CDH *is a converted Diffie–Hellman problem solver to the base G for the group $\mathbb{G}$, then the algorithm* CAIKA *defined by*

$$(Y, U) \leftarrow \mathsf{CDH}(X + f(X)A, B) \tag{6.5.6}$$

$$\mathsf{CAIKA}(A, B, X) \rightarrow (Y, d(U)) \tag{6.5.7}$$

is a computational active implicit key authentication adversary with

- *success rate* $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{CDH})$, *and*
- *computational cost* $\gamma(\mathsf{CAIKA}) \leqslant \gamma(\mathsf{CDH}) + \gamma(d) + \gamma(f) + (1 + 2\log_2(n))\gamma(\mathbb{G})$.

Proof. Algorithm CAIKA above invokes algorithm CDH, evaluates functions f and d, and does one scalar multiplication and one addition in the group $\mathbb{G}$, so the cost $\gamma(\mathsf{CAIKA})$ is as claimed.

The success indicator for the algorithm CAIKA, according to Definition 4.2.1 will be

$$d(U) \stackrel{\circ}{=} \mathsf{mqv}(A, B, X, Y) \tag{6.5.8}$$

The two inputs to CDH, namely $X + f(X)A, B$ are uniformly and independently distributed in $\mathbb{G}$, in accordance with the process for CDH (Definition 5.6.6), because the A and B are independently and uniformly distributed in $\mathbb{G}$, and A is independent of X, under the process for CAIKA (Definition 4.2.1). Therefore, with probability $\sigma(\mathsf{CDH})$, the it will hold that $U = (X + f(X)A) \otimes (Y + f(Y)B)$. In this event, $d(U) = \mathsf{mqv}(A, B, X, Y)$ and the success indicator for CAIKA, as given by (6.5.8), evaluates to 1. Therefore, $\sigma(\mathsf{CAIKA}) \geqslant \sigma(\mathsf{CDH})$. □

6.6 Necessary Joint Conditions on the Group and Key Derivation Function

6.6.1 Derived Diffie–Hellman Problem Hard

The derived Diffie–Hellman problem (Definition 5.5.1) must be hard, or else MQV would fall to a computational passive implicit key authentication attack (Definition 4.1.1), as in the following lemma.

Lemma 6.6.1. *If* KDH *is a derived Diffie–Hellman problem solver, then the algorithm* CPIKA *defined by*

$$\mathsf{CPIKA}(A, B, X, Y) \to \mathsf{KDH}(X + f(X)A, Y + f(Y)B) \tag{6.6.1}$$

is computational passive implicit key authentication adversary for MQV with

- *success rate* $\sigma(\mathsf{CPIKA}) = \sigma(\mathsf{KDH})$, *and*
- *computational cost* $\gamma(\mathsf{CPIKA}) \leqslant \gamma(\mathsf{KDH}) + 2\gamma(f) + (2 + 4\log_2(n))\gamma(\mathbb{G})$.

Proof. Algorithm CPIKA, from (6.6.1) above, executes algorithm KDH, evaluates f twice, and does two scalar multiplication and two additions in $\mathbb{G}$, so the computational cost $\gamma(\mathsf{CPIKA})$ is as claimed.

The success indicator from Definition 4.1.1 for this adversary CPIKA works out to be

$$\mathsf{KDH}(X + f(X)A, Y + f(Y)B) \stackrel{\circ}{=} \mathsf{mqv}(A, B, X, Y) \tag{6.6.2}$$

Inputs $P = X + f(X)A$ and $Q = Y + f(Y)B$ to algorithm KDH are independently and uniformly distributed in $\mathbb{G}$, in accordance with Definition 5.5.1, because A and B are, and A and B are independent of X and Y, under Definition 4.1.1.

Let k be the output of KDH, when invoked by CPIKA. By definition of the function mqv, we have $\mathsf{mqv}(A, B, X, Y) = d(P \otimes Q)$. Therefore, (6.6.2) is $k \stackrel{\circ}{=} d(P \otimes Q)$. So, CPIKA and KDH have the same success indicator. By above, the inputs provided to KDH are in accordance to its process. Therefore, the expected value of the success indicators is the same, $\sigma(\mathsf{CPIKA}) = \sigma(\mathsf{KDH})$. □

6.6.2 Decisional Derived Diffie–Hellman Problem Hard

The decisional derived Diffie–Hellman problem (Definition 5.5.12) must be hard, or else Menezes–Qu–Vanstone would be subject to a decisional passive implicit key authentication attack (Definition 4.3.1), as in the following lemma.

Lemma 6.6.2. *If* DKD *is a decisional derived Diffie–Hellman problem solver, then algorithm* DPIKA *defined by*

$$\mathsf{DPIKA}(A, B, X, Y, k) \to \mathsf{DKD}(X + f(X)A, Y + f(Y)B, k) \tag{6.6.3}$$

is a decisional passive implicit authentication adversary with

- *success rate* $\sigma(\mathsf{DPIKA}) \geqslant \sigma(\mathsf{DKD})$, *and*
- *computational cost* $\gamma(\mathsf{DPIKA}) = \gamma(\mathsf{DKD}) + 2\gamma(f) + 4\log_2(n)\gamma(\mathbb{G})$.

Proof. Algorithm DPIKA from (6.6.3) above invokes algorithm DKD, evaluates function f twice, and does two scalar multiplications in $\mathbb{G}$, so its computational cost $\gamma(\mathsf{DPIKA})$ has the claimed upper bound.

Under the process for DPIKA (Definition 4.3.1), elements A, B are uniformly and independently distributed in $\mathbb{G}$. Let $P = X + f(X)A$ and $Q = Y + f(Y)B$. Because f does not take value 0, element P is uniformly distributed in $\mathbb{G}$. Similarly, Q is uniformly distributed, and independently of P.

The definition of P and Q above ensure that $d(P \otimes Q) = \mathsf{mqv}(A, B, X, Y)$. Under the process for DPIKA, session key k is distributed as $\mathbb{K} \vee \mathsf{mqv}(A, B, X, Y)$. The distribution of k is equivalent to $\mathbb{K} \vee d(P \otimes Q)$, the required distribution for k under the process for DKD.

The success indicators for DPIKA and DKD are equal, and the DKD inputs are distributed in accordance with its process if the inputs of DPIKA are, so $\sigma(\mathsf{DPIKA}) = \sigma(\mathsf{DKD})$. □

6.7 Necessary Security Conditions for Each Grade

This section takes, for each grade of implicit key authentication, the logical conjunction of the all the conditions that have been identified in this report as necessary for that grade, and formalizes results into a corollary.

6.7.1 Necessary Conditions for Computational Passive Implicit Key Authentication

Corollary 6.7.1. *For Menezes–Qu–Vanstone key agreement to provide computational passive implicit key authentication, a necessary condition is that the derived Diffie–Hellman problem hard.*

Proof. The condition is necessary because KDH ⇒ CPIKA from Lemma 6.6.1. □

6.7.2 Necessary Conditions for Computational Active Implicit Key Authentication

Corollary 6.7.2. *For Menezes–Qu–Vanstone key agreement to provide computational active implicit key authentication, some necessary conditions are that*

1. *the derived Diffie–Hellman problem is hard, and*
2. *the converted Diffie–Hellman problem is hard.*

Proof. The first condition is necessary because KDH ⇒ CPIKA from Lemma 6.6.1 and CPIKA ⇒ CAIKA from Proposition 4.5.1. The second condition is necessary because CDH ⇒ CAIKA from Lemma 6.5.3. □

6.7.3 Necessary Conditions for Decisional Passive Implicit Key Authentication

Corollary 6.7.3. *For Menezes–Qu–Vanstone key agreement to provide decisional passive implicit key authentication, a necessary condition is that the decisional derived Diffie–Hellman problem is hard.*

Proof. The condition is necessary because DKD ⇒ DPIKA from Lemma 6.6.2. □

6.7.4 Necessary Conditions for Decisional Active Implicit Key Authentication

Corollary 6.7.4. *For Menezes–Qu–Vanstone key agreement to provide decisional active implicit key authentication, some necessary conditions are that*

1. *the decisional derived Diffie–Hellman problem is hard, and*
2. *the converted Diffie–Hellman problem is hard.*

Proof. The first condition is necessary because DKD ⇒ DPIKA from Lemma 6.6.2 and DPIKA ⇒ DAIKA from Proposition 4.5.2.

The second condition is necessary because CDH ⇒ CAIKA from Lemma 6.5.3 and CAIKA ⇒ DAIKA from Proposition 4.5.4. □

Remark 6.7.1. In other words, the condition identified as necessary for Menezes–Qu–Vanstone to provide decisional active implicit key authentication is a merely separable condition consisting of a conjunctive compound. Further research may find other conditions.

7 Sufficiency of Certain Conditions on MQV Components for IKA

In this section, certain conditions are proved sufficient to Menezes–Qu–Vanstone to provide various grades of implicit key authentication.

7.1 Lesser Forms of Sufficiency

A partially-sufficient condition is such that it and another condition are sufficient. In other words, a partially-sufficient condition is a constituent condition of the sufficient compound condition, where the compound is a logical conjunction.

7.2 Computational Passive IKA

If the derived Diffie–Hellman problem is hard, then Menezes–Qu–Vanstone key agreement provides computational passive implicit key authentication, because of the following theorem.

Theorem 7.2.1. *If* CPIKA *is a computational passive implicit key authentication adversary and the conversion function is never zero valued, then the algorithm* KDH *defined by*

$$X, Y \leftarrow \mathbb{G} \tag{7.2.1}$$

$$\mathsf{KDH}(P, Q) \to \mathsf{CPIKA}(f(X)^{-1}(P - X), f(Y)^{-1}(Q - Y), X, Y) \tag{7.2.2}$$

is a derived Diffie–Hellman problem solver with

- *success rate* $\sigma(\mathsf{KDH}) = \sigma(\mathsf{CPIKA})$, *and*
- *computation cost* $\gamma(\mathsf{KDH}) \leqslant \gamma(\mathsf{CPIKA}) + 2\gamma(f) + (2 + 8\log_2(n))\gamma(\mathbb{G}) + 4\log_2(n)\gamma(\$)$.

Proof. Algorithm KDH above executes CPIKA, evaluates f twice, does two scalar multiplications and two subtractions in $\mathbb{G}$, and generates two random elements in $\mathbb{G}$, which can done by selecting two random elements in $\mathbb{Z}_n$ and two further scalar multiplications in $\mathbb{G}$. (The inversions modulo n are ignored in the cost.) Therefore, the cost $\gamma(\mathsf{KDH})$ is as stated above.

In algorithm KDH above, let $A = f(X)^{-1}(P - X)$ and $B = f(Y)^{-1}(Q - Y)$. With this notation we have,

$$\begin{aligned} \mathsf{mqv}(A, B, X, Y) &= d((X + f(X)A) \otimes (Y + f(Y)B)) \\ &= d((X + f(X)(f(X)^{-1}(P - X))) \otimes (Y + f(Y)(f(Y)^{-1}(Q - Y)))) \\ &= d(P \otimes Q), \end{aligned} \tag{7.2.3}$$

The success indicator in the general process (Definition 5.5.1) for KDH is computed as $(k \stackrel{?}{=} d(P \otimes Q))$ where k is the output of $\mathsf{KDH}(P, Q)$. The success indicator computed in the general process (Definition 4.1.1) for CPIKA is computed $k \stackrel{?}{=} \mathsf{mqv}(A, B, X, Y)$ where k is the output of $\mathsf{CPIKA}(A, B, X, Y)$. Both values of k are the CPIKA above, because the output of CPIKA is defined to be the output of KDH. Equation 7.2.3 says $d(P \otimes Q) = \mathsf{mqv}(A, B, X, Y)$. So the success indicator of KDH equals the success indicator of CPIKA.

In the general process for KDH, variables P and Q are distributed uniformly in $\mathbb{G}$. Because P and Q are uniform in $\mathbb{G}$, so are A and B. In the general process for CPIKA, the four inputs are expected to be uniformly and independently distributed in $\mathbb{G}$. Algorithm KDH provides four uniformly and independently distributed inputs A, B, X, Y, as required for the process of CPIKA. Therefore the expected value of the success indicator for CPIKA, as used in KDH above, is $\sigma(\mathsf{CPIKA})$.

Since our KDH has the same success indicator as the CPIKA it invokes, it has the same expected value of success indicator. Therefore, $\sigma(\mathsf{KDH}) = \sigma(\mathsf{CPIKA})$. □

Remark 7.2.1. A hard derived Diffie–Hellman problem is also necessary condition. So, a hard derived Diffie–Hellman problem is formally equivalent to MQV providing computational passive implicit key authentication.

Remark 7.2.2. Theorem 7.2.1 relies on the output of f being invertible (nonzero) in $\mathbb{Z}_n$. This reliance is merely for simplicity of the proof. If f has high probability of taking value 0, then the algorithm $\mathsf{CFP}() \to 0$, is a high-success conversion function predictor, and by Lemma 6.3.1, there would be a high-success CAIKA algorithm. If one is unconcerned about a CAIKA algorithm, then one may allow f to have high probability of yielding 0. Then Theorem 7.2.1 can be amended as follows. The algorithm is

$$r \leftarrow \mathbb{Z}_n \tag{7.2.4}$$
$$X \leftarrow rP \tag{7.2.5}$$
$$Y \leftarrow r^{-1}Q \tag{7.2.6}$$
$$k \leftarrow \mathsf{CPIKA}(f(X)^{-1}(P - X), f(Y)^{-1}(Q - Y), X, Y) \tag{7.2.7}$$
$$U, V \leftarrow \mathbb{G} \tag{7.2.8}$$
$$l \leftarrow \mathsf{CPIKA}(U, V, X, Y) \tag{7.2.9}$$
$$\mathsf{KDH}(P, Q) \to k \vee l \tag{7.2.10}$$

If both $f(X)$ and $f(Y)$ are nonzero, then k is well-defined. Otherwise k is not defined. One can show $\sigma(\mathsf{KDH}) \geqslant \frac{1}{4}\sigma(\mathsf{CPIKA})$. (At least half the time either both $f(X)$ and $f(Y)$ are nonzero or both are zero. In this event, one of the keys k and l has probability $\sigma(\mathsf{CPIKA})$ of being $d(P \otimes Q)$. With probability $\frac{1}{2}$, algorithm KDH chooses the correct key.)

Remark 7.2.3. Theorem 7.2.1 can easily be interpreted as saying that the computational passive implicit key authentication provided by Menezes–Qu–Vanstone is equivalent to that the provided by static Diffie–Hellman key agreement.

Indeed, as noted §4.6.3, with respect to passive attacks, Menezes–Qu–Vanstone can be expected to be just like authenticated Diffie–Hellman key agreement. So the formal equivalence should not be a surprise.

Remark 7.2.4. Remark 7.2.3 might appear flawed because of the following difference in security between static Diffie–Hellman key agreement and Menezes–Qu–Vanstone key agreement.

If key derivation reversal is easy, then static Diffie–Hellman becomes subject to the Brown–Gallant and Cheon attacks, whereas Menezes–Qu–Vanstone does not appear to be vulnerable, by virtue of the fresh ephemeral private keys (see Remark 5.2.16).

The reason the attack above does not contradict Remark 7.2.3 is that the attack above is not an implicit key authentication attack, but rather a known-key attack, because it presumes that an attacker has access to the agreed session key k.

7.3 Decisional Passive IKA

If the decisional derived Diffie–Hellman problem (Definition 5.5.12) is hard, then Menezes–Qu–Vanstone key agreement provides decisional passive implicit key authentication (Definition 4.3.1), because of the following lemma.

Theorem 7.3.1. *If* DPIKA *is a decisional passive implicit key authentication adversary and f never takes*

value 0*, then the algorithm* DKD *defined by*

$$X, Y \leftarrow \mathbb{G} \tag{7.3.1}$$

$$\mathsf{DKD}(P, Q, k) \rightarrow \mathsf{DPIKA}(f(X)^{-1}(P - X), f(Y)^{-1}(Q - Y), X, Y, k) \tag{7.3.2}$$

is a decisional derived Diffie–Hellman problem solver, with

- *success rate* $\sigma(\mathsf{DKD}) \geqslant \sigma(\mathsf{DPIKA})$.
- *computational cost* $\gamma(\mathsf{DKD}) \leqslant \gamma(\mathsf{DPIKA}) + 2\gamma(f) + (2 + 8\log_2(n))\gamma(\mathbb{G}) + (4\log_2(n))\gamma(\mathbb{G})$

Proof. Algorithm DKD above invokes algorithm DPIKA, evaluates function f twice, generates two random elements in $\mathbb{G}$ (which can each be done with cost at most $2\log_2(n)(\gamma(\$) + \gamma(\mathbb{G}))$, does two scalar multiplications and two group operations in $\mathbb{G}$, and does inversions in $\mathbb{Z}_n$ (which will not be counted in the cost), so the computational cost $\gamma(\mathsf{DKD})$ has the claimed upper bound.

In the process for DKD (Definition 5.5.12), elements P and Q are uniformly and independently distributed in $\mathbb{G}$. Let $A = f(X)^{-1}(P - X)$ and $B = f(Y)^{-1}(Q - Y)$. The resulting four elements A, B, X, Y are uniformly and independently distributed in $\mathbb{G}$, which is in accordance with the process for DPIKA (Definition 4.3.1). In the process for DKD, the session key is distributed as $k \leftarrow \mathbb{K} \vee d(P \otimes Q)$.

By the definition of A and B above, $d(P \otimes Q) = \mathsf{mqv}(A, B, X, Y)$. In accordance with the process for DPIKA, the session key is distributed as $k \leftarrow \mathbb{K} \vee \mathsf{mqv}(A, B, X, Y)$. The expected value of the success indicator for DPIKA is $\sigma(\mathsf{DPIKA})$.

The success indicator for DKD equals the success indicator for DPIKA, because DKD outputs the output bit b from DPIKA and the $d(P \otimes Q) = \mathsf{mqv}(A, B, X, Y)$. So, $\sigma(\mathsf{DKD}) = \sigma(\mathsf{DPIKA})$. □

7.4 Computational Active IKA – To Be Completed

To be completed.

7.5 Decisional Active IKA – To Be Completed

To be completed.

8 Conclusion

To be completed.

Static Diffie–Hellman key agreement lacks some secondary security objectives of a key agreement scheme, such as forward secrecy. Authenticated ephemeral Diffie–Hellman key agreement may provide some of these second security objectives, such as forward secrecy, but extra steps are needed to authenticate the ephemeral keys, such as using digital signatures.

Menezes–Qu–Vanstone key agreement combines static and ephemeral keys without digital signatures. On one hand, the lack of digital signatures improves efficiency, key separation, and plausible deniability. On the other hand, the way it combines ephemeral and static keys opens up the possibility of new attacks on the primary security objective of key agreement (implicit key authentication). More specifically, unlike static Diffie–Hellman key agreement, an adversary can modify the ephemeral public keys. Unlike, signed ephemeral Diffie–Hellman key agreement, an implicit key authentication attack does not seem to be related in any way to a signature forgery, so that that the security of Menezes–Qu–Vanstone does not seem to be able directly rely upon the security of a digital signature scheme such as ECDSA.

This report examined such threats against implicit key authentication, and found that, for implicit key authentication, Menezes–Qu–Vanstone indeed relies upon conditions, such as the hardness of the one-up problem, that authenticated static Diffie–Hellman does not seem to rely upon. If, the one-up problem turns out to be easier than the Diffie–Hellman problem, then the Menezes–Qu–Vanstone would provide less implicit key authentication than static Diffie–Hellman. (But even in this event, Menezes–Qu–Vanstone may still be able to provide secondary security objectives that static Diffie–Hellman key agreement cannot provide.)

Remark 8.0.1. Variants of the one-up problem have been identified earlier. Both ECDSA and ECOH rely upon variants of the one-up problem.

A conjecture that the one-up problem is as hard as the Diffie–Hellman problem appears to remove this discrepancy. Perhaps further work is needed.

To be completed.

References

[ABM+03] A. Antipa, D. R. L. Brown, A. J. Menezes, R. Struik and S. A. Vanstone. *Validation of elliptic curve public keys*. In Y. G. Desmedt (ed.), *Public Key Cryptography — PKC 2003*, Lecture Notes in Computer Science, vol. 2567, pp. 211–223. IACR, Springer, Jan. 2003.

[AS11] E. Andreeva and M. Stam. *The symbiosis between collision and preimage resistance*. In L. Chen (ed.), *Coding and Cryptography – IMACC 2011*, Lecture Notes in Computer Science, vol. 7089, pp. 152–171. Dec. 2011.

[Bro08] D. R. L. Brown. *One-up problem for (EC)DSA*. ePrint 2008/286, IACR, 2008.

[Kra05] H. Krawczyk. *HMQV: A high-performance secure Diffie–Hellman protocol*. ePrint 2005/176, IACR, 2005.

[KJP06] S. Kunz-Jacques and D. Pointcheval. *About the security of MTI/C0 and MQV*. In R. D. Prisco and M. Yung (eds.), *Security and Cryptography for Networks—SCN 2006*, no. 4116 in Lecture Notes in Computer Science, pp. 156–172. Springer, Sep. 2006.

[LMQ+98] L. Law, A. J. Menezes, M. Qu, J. A. Solinas and S. A. Vanstone. *An efficient protocol for authenticated key agreement*. CORR 98-05, Centre for Applied Cryptographic Research, Mar. 1998. Preliminary version of [LMQ+03] at http://www.cacr.math.uwaterloo.ca/techreports/1998/corr98-05.pdf.

[LMQ+03] ———. *An efficient protocol for authenticated key agreement*. Designs, Codes and Cryptography, **28**:119–134, 2003.

[LS02] P. J. Leadbitter and N. P. Smart. *Cryptanalysis of MQV with partially known nonces*. ePrint 2002/145, IACR, 2002.

[LS03] ———. *Analysis of the insecurity of ECMQV with partially known nonces*. In *Information Security — ISC 2003*, Lecture Notes in Computer Science, vol. 2851, pp. 240–251. 2003.

[Nec94] V. I. Nechaev. *Complexity of a determinate algorithm for the discrete logarithm*. Mathematical Notes, **55**(2):165–172, 1994.

[RBB01] P. Rogaway, M. Bellare and D. Boneh. *Evaluation of security level of cryptography ECMQVS (from SEC 1)*. Tech. Rep. 1069, CRYPTREC, Jan. 2001. http://www.ipa.go.jp/security/enc/CRYPTREC/fy15/doc/1069_ks-ecmqv.pdf.

[RS04] P. Rogaway and T. Shrimpton. *Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance*. In B. Roy and W. Meier (eds.), *Fast Software Encryption — FSE 2004*, Lecture Notes in Computer Science, vol. 3017, pp. 371–388. Feb. 2004. See also http://eprint.iacr.org/2004/035.

[Sho97] V. Shoup. *Lower bounds for discrete logarithms and related problems*. In W. Fumy (ed.), *Advances in Cryptology — EUROCRYPT 1997*, no. 1233 in Lecture Notes in Computer Science, pp. 256–266. IACR, Springer, Apr. 1997.

[Wan02] Y. Wang. *Internal report on mqv security in the generic group model.* Tech. rep., Certicom Research, 2002.

A Other Considerations

A.1 Details of Menezes–Qu–Vanstone

The following subsections describe Menezes–Qu–Vanstone key agreement and its components in greater detail. This level of detail is needed to instantiate the componenents of Menezes–Qu–Vanstone key agreement. The instantiations used in standards are described, as well as a few other degenerate instantiations.

Most of this report concerns Menezes–Qu–Vanstone at a higher level of abstraction, with the goal of establishing what criteria of the Menezes–Qu–Vanstone components are needed to imply implict key authentication.

Therefore, the components described in this section can be studied to determine whether they satisfy the criteria listed in this report.

A.1.1 The Group

Recall that Menezes–Qu–Vanstone component $\mathbb{G}$ is a group of prime order n. Because $\mathbb{G}$ has prime order, it is a cyclic group. Because it is cyclic, it is abelian (commutative). Because $\mathbb{G}$ is abelian, and because MQV is most often used with elliptic curve groups, additive notation will be used for the group operation. Because $\mathbb{G}$ is cyclic, it has a generator. Let G be one such generator for $\mathbb{G}$. The choice of the generator is part of the group component defining Menezes–Qu–Vanstone key agreement.

Generally, elements of $\mathbb{G}$ will be denoted by upper case letters, whereas integer variables and other values will be represented by lower case letters. Integer multiples of group elements (powers in multiplicative notation) are notated by placing the integer multiplier to the left of the group element. For example, $3G = G + G + G$. For every element $A \in \mathbb{G}$, there exists an integer a such that $A = aG$, because $\mathbb{G}$ is cyclic. Because of the additive notation, the identity element of the group will be written as 0. The integer a is called a discrete logarithm[20] of A to the base G. Integer a can be chosen such that $0 \leqslant a < n$, and this report will generally assume all integers are in this range.

Remark A.1.1. Because the group $\mathbb{G}$, and its generator G, are public values shared between Alice and Bob, and therefore between any two parties in a domain of parties who might use MQV key agreement, the values $\mathbb{G}$ and G are parts of what is often called *domain parameters*.

Remark A.1.2. Standardized MQV assumes that $\mathbb{G}$ is either

- A prime order subgroup of an elliptic curve group defined over a finite field (additive notation, much like the notation in this report, is often used for such elliptic curve groups),
- A prime order subgroup of the multiplicative group of a finite field (multiplication notation is almost always used when describing these groups specifically).

These are the most commonly used groups for cryptographic schemes that require a hard discrete logarithm problem.

Remark A.1.3. This report's notation 0 for the identity element of $\mathbb{G}$ is slightly different from the usual notation for the identity element of the specific groups from Remark A.1.2. In elliptic curve groups, the notation $\mathcal{O}$ is often used. In multiplicative groups of finite fields, because of the multiplicative notation, 1 is used.

In multiplicative groups of finite fields, further cause of confusion may arise due to the presence of a field, which also uses additive notation for a different group. In particular, 0 in this report's notation does not refer to the zero element of the field, since this is outside the group $\mathbb{G}$.

[20] The terminology originates from the original Diffie–Hellman groups; they typically have multiplicative notation.

A.1.2 Static Public Key Generation and Distribution

A party, say Alice, who wishes to participate in Menezes–Qu–Vanstone key agreement needs to arrange to have public key pair, whose public key can be trusted by any other party, say Bob, with whom Alice will want to use MQV to agree upon a session key.

Static Private Key Generation First, Alice must generate a private key $a \in \mathbb{Z}_n$. Ideally, Alice will choose a uniformly at random in $\mathbb{Z}_n$. This report will generally assume that Alice will do this, as a simplification to the analysis. Alice must generate and maintain a secretly, so that an adversary Eve will not learn anything about a that could lead to an attack.

Remark A.1.4. In practice, it may suffice for Alice to choose a such that, first, a has at least $\frac{1}{2}\log_2(n)$ bits of min-entropy, and, second, a is computationally indistinguishable from a chosen uniformly at random.

Remark A.1.5. The indistinguishability of Alice's static private key may not be actually necessary, because no general result is known that converts an arbitrary distinguishing algorithm on a into an attack on Menezes–Qu–Vanstone.

Indeed, Schnorr has argued that, in Shoup's generic group model, private keys can be highly distinguishable, without decreasing the difficulty of the discrete logarithm problem. This suggests that it is quite possible for cryptographic schemes relying the discrete logarithm problem to be secure even when the private keys are easily distinguishable from uniform.

Remark A.1.6. Certain specific forms of bias in a, such as a being small, are necessary in order to avoid attacks that allow an adversary to solve discrete logarithms faster, such as Pollard's kangaroo algorithm. These biases can be achieved by selecting a uniformly at random, or indistinguishably from uniform.

Remark A.1.7. Remarks A.1.5 and A.1.6 suggest that pseudorandom private keys are partially sufficient, but may not be necessary. In other words, a gap exists between the known necessary and sufficient conditions with respect to the private keys.

Static Public Key Computation Second, after Alice has generated her static private key a, she computes her public key A by the scalar multiplication $A = aG$ in the group $\mathbb{G}$ to the base G.

Authentic Transmission of the Static Public Key Third, Alice must undertake some arrangements to ensure that Bob will be certain that A belongs to Alice. Alice may deliver A personally to Bob. More typically, Alice may contact a certification authority, who will digitally sign Alice's public key A to give her a certificate.

Authentic Reception of the Static Public Key Bob must use an authentic communication channel to obtain Alice's public key A, or to obtain the public key of the certification authority which he can use to verify the digital certificate in Alice's certificate. Because authentic channels are slow, and also expensive or unreliable, Alice and Bob typically pre-arrange the authentic channel, before commencing with key agreement.

Mirrored Actions of the Other Party Bob has to take the same measures as Alice to generate his own static key pair, a static private key b and a static public key $B = bG$. Alice needs an authentic channel to receive Bob's public key, or the certification authority's public key.

Readiness for Ephemeral Key Exchange Once, Alice and Bob have arranged their static public keys, and the authentication necessary to authenticate each other's public key (typically by obtaining an authentic copy of a certification authority's public key and obtaining a digital certificate of their own public key), they are ready to do Menezes–Qu–Vanstone key agreement.

A.1.3 Ephemeral Public Key Generation and Distribution

Alice generates an ephemeral private key x and an associated ephemeral public key $X = xG$ in the same way that she generates a static key pair, but she does not need to have the public key of the key pair undergo transmission through an authentic channel, either to Bob or to a certification authority.

Alice should generate her ephemeral key pair just before she needs it, because this provides her with forward secrecy. She should also destroy the ephemeral private just after she agrees upon the session key with Bob, for the same reason.

Bob does the same procedure as Alice to generate his ephemeral private key y and associated ephemeral public key $Y = yG$.

Remark A.1.8. Remark A.1.4 about the min-entropy and indistinguishability applies to ephemeral private keys too.

Remark A.1.9. Remark A.1.5 about the apparent non-necessity of indistinguishability applies to ephemeral private keys.

Remark A.1.10. Remark A.1.6 about the risks of bias in private keys is further complicated by the Leadbitter–Smart attack: certain specific biases in the ephemeral keys could potentially compromise the static private keys.

A.1.4 Shared Secret Group Element Calculation

Recall that component f is the conversion function and is a function $f : \mathbb{G} \to \mathbb{Z}_n$.

Remark A.1.11. Standards for MQV define f, in the case of an elliptic curve group, approximately as follows. The integer $f(X)$ is computed by taking x-coordinate of X, representing this finite field element as a bit string, truncating a portion (approximately half) of the bits on left, and prepending with a one-valued bit on the left, and converting the resulting bit string to an integer.

Remark A.1.12. This report may occasionally presume that f is such that $f(X) \not\equiv 0 \bmod n$ for all $X \in \mathbb{G}$. This presumption will allow the writing of $f(X)^{-1} \in \mathbb{Z}_n$. This is true for the f in standardized version of ECMQV from Remark A.1.11.

After selecting a and x, and receiving B and Y, as described above, Alice computes

$$U = (x + f(X)a)(Y + f(Y)B). \tag{A.1.1}$$

Remark A.1.13. Because x, $f(X)$, and a are values in $\mathbb{Z}_n$, value $x + f(X)a$ belongs to $\mathbb{Z}_n$. In other words, all the integer computations done in the expression $x + f(X)a$ are done modulo n.

Remark A.1.14. The value $x + f(X)a$ has sometimes been referred to as Alice's implicit signature. Alice should not reveal her implicit signature to an adversary, or else the adversary could compute her session key.

After selecting b and y, and receiving A and X, as described above, Bob computes

$$V = (y + f(Y)b)(X + f(X)A). \tag{A.1.2}$$

Remark A.1.15. As above, $y + f(Y)b \in \mathbb{Z}_n$, and y is sometimes referred to as Bob's implicit signature, and Bob should not reveal his implicit signature.

If Alice and Bob have generated the static and ephemeral public keys as described above, and also successfully exchanged all of their public keys without interference from an adversary, then $U = V$, because

$$\begin{aligned} U &= (x + f(X)a)(Y + f(Y)B) \\ &= (x + f(X)a)((y + f(Y)b)G) \\ &= ((x + f(X)a)(y + f(Y)b))G \\ &= ((y + f(Y)b)(x + f(X)a))G \\ &= (y + f(Y)b)((x + f(X)a)G) \\ &= (y + f(Y)b)(X + f(X)A) \\ &= V \end{aligned} \tag{A.1.3}$$

where, as above, all integer arithmetic is done modulo n.

Remark A.1.16. The elements $U, V \in \mathbb{G}$ are called the MQV shared secret group elements. The MQV shared secret group elements are not used directly as session keys, because they are only intermediate values, which, like ephemeral private keys, should be destroyed after use.

A.1.5 Session Key Derivation

Recall that the component d is the key derivation function and it is a function $d : \mathbb{G} \to \mathbb{K}$.

Remark A.1.17. This report will presumes that $\mathbb{K}$ is a subset of $\{0,1\}^\kappa$ for some integer κ indicating the bit length of keys used in some symmetric-key algorithm, such as the Advanced Encryption Standard (AES).

The function d is public, but may be individualized to Alice and Bob. Alice and Bob compute their session keys $k = d(U)$ and $l = d(V)$, respectively. When $U = V$, as in equation (A.1.3), Alice and Bob will compute the same session keys $k = d(U) = d(V) = l$.

Remark A.1.18. Standards for elliptic curve Menezes–Qu–Vanstone typically specify $d(U)$ as something like taking a κ-bit truncation of the bit string

$$H(x(hU)\|1\|S)\|H(x(hU)\|2\|S)\|\ldots\|H(x(hU)\|m\|S) \tag{A.1.4}$$

where H is a hash function, such as SHA-1, h is an integer known as the cofactor, $x(hU)$ is a bit string representation of the x-coordinate of the element hU, $\|$ is a concatenation operator, the integers $1, 2, \ldots, m$ are some fixed-bit-length counters, and S is some non-secret data individualized to Alice and Bob.

Remark A.1.19. Standards present the Menezes–Qu–Vanstone key derivation slightly differently from the way presented in this report, by not regarding the scalar multiplication by the cofactor h as part of the key derivation function, but rather an earlier step, as in the shared secret group elements from the previous subsection.

Because n is prime, the first step in the key derivation function of scalar multiplication by h, which sends U to hU, is a efficiently computable and invertible permutation. As such, it will not substantially change the security properties between this report's presentation of the key derivation function and the conventional presentation.

Remark A.1.20. Although the standard MQV key derivation functions have the capacity to produce arbitrarily long bit string session keys, it does not appear to be the intent to use these session keys as one-time pad keystreams. A more typical intent is to use the session key as a key for some symmetric key algorithm (which may still be a stream cipher).

Remark A.1.21. Although standards do not intended long session keys to be used as a one-time pads, per Remark A.1.20, it is possible to include, for the purpose of security definition and analyses a keystream generator, such as AES-CTR or RC4, as part of the formally modeled key derivation function. In this case, the decisional variants of IKA adversaries carry considerably more importance.

Remark A.1.22. If the cofactor h has $h > 1$ and is quadratic residue modulo n, then there exists g such that $g^2 \equiv h \bmod n$. Let $G = gG'$. Then Menezes–Qu–Vanstone key agreement with cofactor h, using a base of generator G, is almost equivalent to Menezes–Qu–Vanstone key agreement with cofactor 1, using a base of generator G', with equivalence in the following sense. No public keys change, although the private keys are adjusted by a factor of g.

The only exception to this equivalence of the transformation above is the the loss of the effect of provided by cofactor multiplication to resist so-called small-subgroup attacks. For the sake of simplifying security analyses in this report, public keys are always assumed to belong to the correct group and to be non-zero, which precludes small-subgroup attacks.

A.2 Alternatives to Menezes–Qu–Vanstone Key Agreement

Many other possible mutually authenticated two-pass key agreement schemes can be compared to Menezes–Qu–Vanstone key agreement. Three typical, standardized schemes are:

1. Static Diffie–Hellman key agreement, where Alice and Bob each have pre-authenticated Diffie–Hellman keys, such as those in certificates. To avoid replay attacks, Alice and Bob may exchange some public nonces so that session keys are different over multiple uses. (The nonces are the two passes.)

2. Signed ephemeral Diffie–Hellman agreement, where Alice and Bob have some pre-authenticated digital signature keys, such as those in certificates. Alice and Bob generated and ephemeral Diffie–Hellman public keys, which they authenticate using digital signatures.

3. Unified model key agreement, which derives session keys from a combination of ephemeral Diffie–Hellman and static Diffie–Hellman.

This report provides formal analysis only for Menezes–Qu–Vanstone key agreement, so the following comparison of Menezes–Qu–Vanstone with other two-pass key agreement schemes remains informal only.

- Static Diffie–Hellman key agreement does not provide forward secrecy and various other secondary security objectives (see §A.3) that Menezes–Qu–Vanstone intends to provide. The lack of forward secrecy is mainly because static Diffie–Hellman does not use any ephemeral private keys.

- Signed ephemeral Diffie–Hellman key agreement generally has greater computational cost than Menezes–Qu–Vanstone key agreement, because the digital signatures generally have greater computational cost the Menezes–Qu–Vanstone shared secret value computations. Furthermore, digital signatures leave third-party verifiable evidence of key agreement between Alice and Bob.

- Unified model key agreement lacks a secondary security objective known as key compromise impersonation. Unified model also has greater computational cost than Menezes–Qu–Vanstone key agreement.

The HMQV key agreement scheme has also been proposed to IEEE P1363 as an improvement MQV key agreement, with the improvements including the domain of provable security.[21]

[21] Showing that some security reduction can also be proved for MQV (so, not just for HMQV) was a partial motivation for this report.

A.3 Secondary Security Objectives Not Studied in this Report

A major advantage of Menezes–Qu–Vanstone key agreement over some other comparable key agreement schemes is its efficiency and potential to provide many secondary security objectives of a key agreement schemes. As noted already above, these secondary security objectives are not studied formally in this report. This report examines only the primary security objective of two-pass key agreement: implicit key authentication. The hope is that Menezes–Qu–Vanstone key agreement's various efficiencies and secondary security benefits do not interfere with the primary security goals.

That Menezes–Qu–Vanstone key agreement is believed to meet its secondary security objective (whereas a static Diffie–Hellman key exchange without the addition of digital signatures, or even Rivest–Shamir–Adleman based key establishment, cannot meet all of these secondary security objectives) justifies the consideration of Menezes–Qu–Vanstone as a key agreement scheme.

Remark A.3.1. The secondary security objectives are now listed informally. Understanding these objectives is not necessary to understanding the results in this report but does help motivate the usefulness of MQV key agreement, and thereby justify why this report studies MQV.

Remark A.3.2. Some of the remarks in this section would be best understood only after reading §2.

Remark A.3.3. Implicit key authentication attacks belong to the class of non-invasive attacks, in which the adversary gets no access to any private keys or session keys. MQV also aims to resist two other non-invasive attacks:

- replay attacks, in which adversary Eve causes Alice and Bob to use a session key that they used previously (Eve does not need to learn the session key, but may still cause harm by causing Alice and Bob by replaying an old message encrypted or authenticated with session key).
- unknown key share (UKS) attacks, in which Alice incorrectly believes that she shares her session key with Eve, whereas she actually shares it with Bob (who correctly thinks he shares it with Alice). Although Eve does not necessarily learn anything about the session key in this type of attack, Eve may still cause harm by making Alice believe that an encrypted, or authenticated, message from Bob is from Eve. For example, if, in a larger protocol which may use MQV, Bob may be expected prove his authorization to Alice by proving possession of a password. If, upon receiving authorization, Alice takes certain actions, then Eve can launch a UKS attack to cause Alice to take unauthorized action. It is questionable whether any larger protocol would need to function in such a way. Kaliski found an UKS attack on certain forms of MQV.

This report does not treat the replay or UKS attacks above.

Remark A.3.4. MQV aims for resistance to some *invasive* attacks, in which Eve is somehow capable of obtaining some normally secret values:

- known-key security attacks, in which Eve obtains some session keys (and then she tries to deduce something about other session keys which she was not able to obtain),
- forward secrecy attacks, in which Eve obtains Alice's static private key (and then Eve tries to learn about Alice and Bob's past session keys, despite not knowing Alice's ephemeral private key or either of Bob's private keys), and
- key compromise impersonation attacks, in which Eve obtains Alice's static private key (and then tries to impersonate Bob by replacing Bob's ephemeral public key with one of her own choosing).

This report does not treat any invasive attacks, including any of those above.

Remark A.3.5. Forward secrecy, which MQV, and other key agreement schemes try to provide, can serve as a partial mitigation to the threat of Shor's algorithm being run on a quantum computer, because it seems to require one

run of Shor's algorithm per session key, rather than per public key. If the threat of Shor's algorithm becomes more realistic, users can choose to use shorter sessions and more frequent uses of key agreement.

Remark A.3.6. This report does not address implementation fault attacks, in which the implementations of Alice and Bob either:

- leak additional information about their private keys (such as the Leadbitter–Smart [LS02, LS03] that extract the static private key from MQV from leaked bits of the ephemeral private key),
- can be fed invalid inputs with invalid formats without throwing an error (similar to the Antipa–Brown–Menezes–Struik–Vanstone [ABM+03] attack on one-pass ECMQV),

Such attacks can be also considered to be invasive.

Remark A.3.7. Agreed session keys are meant to be used. Typically they are used as keys for symmetric-key message encryption and authentication. Such usage leaks a very small amount of information about the session keys, in that they permit an adversary to distinguish between a session key and another key.

If one were to quantify the information leaked, it would be much less than a bit of information, in fact, a negligible fraction of a bit. The main impact of this leaked information is that it permits conversion of low-cost low-success adversaries into high-cost high-success adversaries as noted below.

If an adversary has some candidate values for the session keys, typically a single observation of the usage of the session key can allow the adversary efficiently test each candidate. So, repeated applications of low success rate algorithm for finding the session key can then be converted in a high success rate algorithm.

This trade-off in cost versus success rate may be noted at various points in the report.

Remark A.3.8. The neglible leakage from Remark A.3.7 can perhaps be avoided as follows. Messages would be converted to a non-redundant form, such as by using very strong message compression. The non-redundant (perhaps compressed) message would be encrypted, but not authenticated, because authentication with the agreed session key would certainly allow the above leakage to result.

Generlly, such an approach runs contrary to the prevailing practice, in part because it does not protect the integrity of the message. In other the cost, which is significant, far outweighs the gain, which is negligible.

Remark A.3.9. This report does not place special attention on attacks in which the adversary Eve's success is defined as determining the static private keys (as done in the Leadbitter–Smart attack). Such attacks have sometimes been called *total break* attacks. A non-invasive total break attacker can use the broken private keys to launch an implicit key authentication attack. (The Leadbitter–Smart is an implementation-fault invasive total break attack.) So implicit key authentication provides assurance against non-invasive total break attacks.

If Menezes–Qu–Vanstone key agreement is used in isolation of other cryptographic schemes, then there is not much need for greater protection against non-invasive total break attacks than against implicit key authentication. (Furthermore, even invasive total break attacks correspond to the invasive session key computation or session key distinguishing attacks.) But if the MQV private keys are also used for another purpose, such as for digital signatures, then a total break can cause more harm than the adversary learning an MQV session key. If MQV is used in this way, having even stronger assurance (if possible) against total breaks has value.

Again, this report does not examine if MQV has even greater assurance against total break attacks than it has against implicit key authentication attacks. Regardless, for many other reasons, it is not prudent to use MQV private keys for significantly different purposes, especially for digital signatures.

Remark A.3.10. The definition of implicit key authentication in this report models an adversary who is able to observe every ephemeral key exchanged. When key agreement transmissions are rare and also interspersed among other abundant transmissions, then observing ephemeral keys has a potentially significant cost.

This would suggest to consider a weakened form of adversary who does not get to see the ephemeral keys. Such an adversary is not formalized in this report. Informally speaking, MQV seems to resist such an adversary because

the agreed session key depends on the ephemeral keys to the extent that if they are unseen, they serve much like one-time pads. Again, this report does not examine this issue formally.

By contrast, authenticated static Diffie–Hellman key exchange does not resist this attack.

Remark A.3.11. This report assumes that certain keys are selected uniformly at random. It would seem reasonable that many of the results could be extended to keys that are computationally indistinguishable from uniformly random keys.

Perhaps some of the results could be extended, or adapted, to keys that may be easily distinguishable from uniformly distributed keys, but could not be easily guessed, together with some further assumptions such as those necessary to avoid the Leadbitter–Smart attack [LS02, LS03].

Remark A.3.12. Extra information can be exchanged between Alice and Bob with basic two-pass MQV exchange to provide key confirmation. For two-sided key confirmation, a third pass is needed. Key confirmation includes a modification of two-pass MQV, and also provides a secondary security objective called explicit key authentication, so is out of scope of this report for two reasons. An informal reason to consider explicit key confirmation as secondary is that the agreed keys should only used in secure algorithms, which really should robust enough to be used straight away without confirming the other party has the key. That is, it suffice that the other party is the only one could possibly know the key. An arguable benefit to key confirmation is to prevent denial-of-service attacks, but there are many other ways to achieve denial-of-service, and also better ways than explicit key confirmation to mitigate against denial-of-service attacks.

Remark A.3.13. As noted earlier, MQV key agreement does not use digital signatures for authentication. Neither Alice nor Bob can provide to the third parties the others' signature as evidence of participation in the key agreement. This aspect of a key agreement scheme has been called plausible deniability, or even off-the-record communication.

A.4 Motivation for, and Approach of, this Report

Matt Campagna[22] asked me to provide a security proof for ECMQV composed with ECQV implicit certificates, which may have helped promote a standardization effort to use ECMQV with ECQV.

Remark A.4.1. This request also happened soon after ECMQV was removed from Suite B. All indications were that ECMQV was removed for reasons of interoperability. Nevertheless, Campagna raised the remote possibility that a security flaw was the reason for the removal. More likely was the possibility that others could claim that a security flaw was the reason, even if it was not. Conducting a security review, with possible security proof, seemed a prudent way to alleviate these concerns.

Remark A.4.2. ECMQV has been, and still is, being used to protect the enterprise communications on millions of BlackBerry smartphones. A careful security analysis is warranted for this reason alone. At best, one can hope for any additional security re-assurance, over and above the existing assurances; at worst, one can realize that certain problems need to be corrected.

Remark A.4.3. At the time, unaware of any security proofs for ECMQV, I deemed that a more manageable first step in such a project would be to find a security proof for ECMQV alone, rather than its composition with ECQV. Being unfamiliar with key agreement, I opted to examine only the most essential security aspects of key agreement, namely implicit key authentication.

I decided to forgo both the random oracle model and generic group model, even though both these models should have made proving security easier. I avoided the random oracle model partly because it seemed like overkill, and because I perceived that the key derivation functions warranted more scrutiny. Also, proofs not using random oracles have lately been given more value. I avoided the generic group model because I was already aware of work done by Yongge Wang using that model.

[22] My former manager.

Having a research background in reductions, for both necessary and sufficient conditions, I decided to continue with that approach.

Not having an any ingenius insights, I chose to seek straightforward reductions: those which forward the results of one algorithm to another, only filling any missing variable at random, or completing any missing calculations in accordance with an ordinary user's calculations in Menezes–Qu–Vanstone.

Being straightforward, most reductions of this nature that I found were obvious, and many essentially well-known when their generality went beyond MQV. The reductions also entailed devising various definitions of security for the components of MQV. This had to be chosen to fit the approach taken in this report, but many ended be very similar to conventional notions for these component, not surprisingly.

A.5 MQV as DH via Proxy Protocol – To Be Completed

To be completed.

Although it is customary to view MQV as an alternative improvement to DH, it is also possible to interpret MQV key agreement as an instance of ephemeral DH key agreement, with an extra authentication step.

Further it is possible for Alice and Bob to communicate via proxies Polly and Quentin, respectively. Alice and Bob do MQV. Their Polly and Quentin externally ephemeral DH, with some unconventional type of authentication on top.

Alice has static public key $A = aG$ and generates ephemeral public key $X = xG$. A proxy can run in three modes: private, session, and public.

- If Polly is a private proxy, then Alice sends $p = x + f(X)A$ to Polly. Polly can compute the effective ephemeral public key $P = pG$.
- If Polly is a session proxy, then Alice sends X to Polly. Polly computes $P = X + f(X)A$.
- If Polly is a public proxy, then Alice sends X to Polly, and Polly computes $P = X + f(X)A$.

In all three modes, Polly has an effective ephemeral DH public key P, and sends this Quentin.

If Polly is a public proxy, then Polly also sends X to Quentin.

When Quentin receives P from Polly, his behavior depends on his mode.

- If Quentin is a private proxy, then he will have received $q = y + g(Y)b$ from Bob, which Quentin then uses this as private ephemeral private key. Quentin can then compute $l = d(qP)$.
- If Quentin is a session proxy, Quentin forwards P to Bob, and Bob computes $l = d((y + f(Y)b))P)$. After Bob compute l, he forwards l to Quentin.
- If Quentin is a public proxy, and Polly is too, then he will receive P and X and forward X to Bob.

The proxies also convey information in the opposite direction.

If Polly and Quentin are public proxies, then Alice and Bob do their usual calculations, and the information exchanged between the proxies are the pairs (P, X) and (Q, Y).

If Polly and Quentin are private or session proxies, then Polly and Quentin can compute the session key, which they can use to create an anonymous but confidential channel. They encrypt the X and Y values to send them over the encrypted channel. This encryption may provide some extra protections, such as better anonymity.

If Quentin is a session proxy, then Quentin forwards X to Bob. Bob confirms confirms that $P = X + f(X)A$, and thereby confirms that he has done key agreement with Alice.

If Quentin is a private proxy, then Quentin can ...

In all three modes, the proxy protocol between Polly and Quentin doubles the cryptographic payload, because it explicitly transmits the values P and Q, which normally do not have to be made transmitted in MQV. Also, proxy protocol bears some resemblance to other Diffie–Hellman protocols, such as those that sign, or otherwise authenticate, the ephemeral public DH keys.

When Bob receives X, he can check its consistency with P and A, by checking that $P = X + f(X)A$.