Prime, Order Please!
Revisiting Small Subgroup and Invalid Curve Attacks on Protocols using Diffie-Hellman

Cas Cremers
CISPA Helmholtz Center for Information Security
Saarland Informatics Campus, Germany

Dennis Jackson
University of Oxford
Oxford, United Kingdom

***Abstract*—Diffie-Hellman groups are a widely used component in cryptographic protocols in which a shared secret is needed. These protocols are typically proven to be secure under the assumption they are implemented with prime order Diffie Hellman groups. However, in practice, many implementations either choose to use non-prime order groups for reasons of efficiency, or can be manipulated into operating in non-prime order groups. This leaves a gap between the proofs of protocol security, which assume prime order groups, and the real world implementations. This is not merely a theoretical possibility: many attacks exploiting small subgroups or invalid curve points have been found in the real world.**

While many advances have been made in automated protocol analysis, modern tools such as TAMARIN and ProVerif represent DH groups using an abstraction of prime order groups. This means they, like many cryptographic proofs, may miss practical attacks on real world protocols.

In this work we develop a novel extension of the symbolic model of Diffie-Hellman groups. By more accurately modelling internal group structure, our approach captures many more differences between prime order groups and their actual implementations. The additional behaviours that our models capture are surprisingly diverse, and include not only attacks using small subgroups and invalid curve points, but also a range of proposed mitigation techniques, such as excluding low order elements, single coordinate ladders, and checking the elliptic curve equation. Our models thereby capture a large family of attacks that were previously outside the symbolic model.

We implement our improved models in the TAMARIN prover. We find a new attack on the Secure Scuttlebutt Gossip protocol, independently discover a recent attack on the Tendermint protocol, and show how our analysis finds previous Bluetooth attacks and evaluate the effectiveness of the proposed countermeasures.

I. Introduction

Since the inception of public-key cryptography, the Diffie-Hellman (DH) construction based on the perceived hardness of the discrete logarithm problem [22] has been one of the most common building blocks of secure systems, and underlies most of our secure communications. At its core, the DH construction relies on the hardness of computing the discrete logarithm in a particular group.

Typically, a computational proof that a protocol is secure will make the assumption that the Diffie Hellman group is of prime order. Similarly, modern protocol analysis tools such as ProVerif and TAMARIN, which can automatically construct symbolic proofs or find attacks on protocols, internally use an abstraction of Diffie-Hellman groups that reflects the properties of a prime order group.

In contrast, most modern real-world implementations that use DH-constructions do not use groups of prime order. For example, protocols such as TLS, SSH, and IKE have included support for finite field groups with highly composite order such as DSA groups, or even randomized groups whose properties are difficult to verify. Recently, elliptic curves have seen widespread adoption. Whilst some elliptic curves are of prime order; many popular choices, such as Curve25519, are not.

Thus, while many implementations operate in non-prime order groups, protocol analyses often assume that operations are performed in prime order groups. The result is not just a gap in the analysis: many attacks have been discovered that exploit the additional structure present in non-prime order groups. Prominent examples of such attacks are catastrophic key recovery attacks on TLS, SSH and IKE using finite field groups [54], triple handshake attacks on TLS clients using randomized groups [9], and channel binding attacks on compound authentication protocols running over IKE and SSH [8].

Another assumption implicitly made in proofs of security is that Diffie Hellman group elements can be parsed unambiguously. That is, a correct protocol implementation will reject inputs which do not correspond to group elements. This has been problematic, particularly when Diffie Hellman groups are instantiated with elliptic curves and the consequent attacks have been disastrous: private key recovery attacks on TLS [30], confidentiality and authentication attacks on Bluetooth [11] and key compromise of JSON Web Tokens [21].

These attacks have led to the introduction of various countermeasures, not all of which have been effective. Furthermore, depending on the protocol scenario and underlying structures, it may even be the case that no countermeasures are needed. Thus, the existence of of these subtle attacks and countermeasures motivates the need for including the details of the implemented groups and countermeasures into the security analysis.

In this work we develop a family of novel symbolic models for DH groups for use in modern protocol analysis tools, that enables finding such attacks or symbolically prove their absence. We provide the means for a more fine-grained symbolic analysis that allows the protocol modeller to specify the type of underlying group as well as any countermeasures, thereby closing a gap in the automated analysis of real-world protocols. In the process, we find a new attack and independently discover a recent attack on two real-world protocols.

A. *Contributions*

1) We develop a novel family of symbolic models of Diffie-Hellman groups that captures the internal structure of non-prime order groups used in the real world. This model makes it possible to discover real world attacks, such as attacks based on small subgroup confinement, small subgroup key leakage and invalid elliptic curve points. Additionally, our model supports accurate representations of the various mitigations available to protocol designers, such as the exclusion of low order points, curve equation checks, and single coordinate ladders. This allows for effective symbolic verification of a protocol, despite the use of a non-prime order group.
2) Using the TAMARIN prover, we develop the first automated method for finding or proving the absence of small subgroup and invalid curve point attacks on security protocols that use Diffie Hellman groups. We evaluate its effectiveness on several case studies and show it is effective at both attack finding and verification.
3) In the process, discover a new attack and independently discover a recently found attack on real-world protocols: (i) We discover a new attack on the Secure Scuttlebutt Gossip protocol, which violates a core authentication property, resulting in the disclosure of private information, despite it previously having been verified as secure in a coarser symbolic model. (ii) We independently discover a recently disclosed attack on the Tendermint's secure handshake protocol, which allows for the impersonation of legitimate users.

Outline: We proceed as follows. We first give background in Section II and recap existing Symbolic Diffie-Hellman models in Section III. In Section IV, we present our new symbolic models for three classes of Diffie Hellman groups used in implementations that each add particular attack vectors. These models and behaviours are agnostic as to the underlying realisation of the group, whereas the commonly used Elliptic Curves allow for additional behaviours that can lead to attacks. To capture these, we present in Section V a symbolic model for more accurately capturing Elliptic Curve points. We complete our models in Section VI by introducing models for common mitigations. In Section VII we apply our techniques to several real-world case-studies, explore our results and conclude in Section VIII.

Our source code and models are available from [20].

II. Background

A. *Diffie Hellman Groups*

For more background on group theory, including definitions for the various terms we use here, see appendix C on page 15. We also discuss the mathematics of finite fields and elliptic curves in §D and §E. In this section we recap some elementary theorems from group theory we will be using, as well as defining our notation for Diffie Hellman groups.

There are two common notations for Diffie Hellman groups. When referring to finite fields, typically multiplicative notation is used where we write the group operation as $\cdot$ and describe repeated applications as exponentiation. However, in the elliptic curve setting we use additive notation where the group operation is written $+$ and repeated applications are described as scalar multiplication. *To avoid ambiguity, we use multiplicative notation throughout this paper, regardless of the underlying structure.*

The following well known theorems will also be of use:

Lagrange's Theorem. Let G be a finite group and H a subgroup of G. The order of H divides the order of G. It immediately follows that for $g \in G$, $|g|$ divides $|G|$.

Cauchy's Theorem. Let G be a finite group and p a prime number dividing the order of G. Then G contains an element of order p (and hence a subgroup of order p)

The fundamental theorem of finite abelian groups. Let G be a finite abelian group of order n. Let the unique factorisation of n into distinct prime powers be given by $n = p_1^{a_1} \dots p_k^{a_k}$. Then:

1) $G \cong A_1 \times \dots \times A_k$ where $|A_i| = {p_i}^{a_i}$
2) For each $A \in \{A_1, \dots, A_k\}$ with $|A| = p^a$
$$A \cong \mathbb{Z}_{p^{b_1}} \times \dots \times \mathbb{Z}_{p^{b_t}}$$
with $b_1 \geq b_2 \geq \dots \geq b_t$ and $b_1 + \dots + b_t = a$
3) The decomposition given above is unique.

B. *The Symbolic Model and* TAMARIN

Unbounded symbolic automated protocol analysis tools such as TAMARIN [43], ProVerif [12], accept a formal description of a protocol and its intended security properties, and try to establish a proof that no attack exists within the tool's model, or exhibit a counterexample that violates the security property.

In TAMARIN's framework, the execution of a protocol in an adversarial environment is modelled using a labelled transition system. The state includes network messages, the adversary's knowledge, and the internal protocol state. The protocol and adversary interact by exchanging network messages, which are controlled by the adversary. Both protocol rules and adversary capabilities are specified as labelled multiset rewrite rules that define the valid transitions between system states, forming a trace. Security requirements are specified in a (guarded) fragment of first-order logic with quantification over timepoints.

The transition system state is given by a multiset of *facts*. Facts are symbols that take any (fixed) number of terms as their arguments. There is a special set of facts that encodes network messages as well as the adversary's knowledge. We generally write $\mathit{Factname}(t_1, t_2, t_3)$ for a fact named $\mathit{Factname}$ with three terms as its arguments.

A *labelled multiset rewriting rule* is written:

```
rule name:
[ l ] --[ a ]-> [ r ]
```

where `rule` is a keyword, `name` is an identifier, and `l`, `a`, `r` are multisets of facts representing the *premises*, *actions*, and *conclusions* respectively. All of these may contain variables. Some rules may have no associated action, and in that case we omit the action `[ a ]` in the rule description.

The *labelled transition system* operates on ground terms, that is, terms without variables. A rule is *applicable* to a given ground state when an instantiation of the premises of the rule is contained in said state. Applying a rule removes the instantiated premises, and adds appropriately instantiated conclusions. The instantiated action facts represent the labels of the labelled transition, thereby defining the traces. In particular, an *execution* is a sequence of states starting from the empty set and using *rule instances* to transition from one state to the next. Then, a *trace* is the sequence of ground action facts appearing at the rule instances in a protocol execution.

Security properties are defined in a fragment of two-sorted first-order logic, with messages and timepoints, and quantification is possible over both. The atoms considered are $\perp$, term equality `s = s'`, timepoint ordering $t_1 < t_2$, timepoint equality $t_1 = t_2$, or an action *Fact* at a timepoint t_1 written $\mathit{Fact}(\mathit{terms})@t_1$, together with the usual logical connectives. This allows expressing a wide range of security properties.

We additionally consider *restrictions* which limit the explored state space. Technically, restrictions are formulas just like security properties, and ensure that all valid traces must satisfy the restriction formulas. They are often used to model conditional rewriting rules (such as inequality checks) or properties involving global state. More TAMARIN details can be found at [43].

III. Existing Symbolic Diffie Hellman Models

We now discuss existing symbolic models of Diffie Hellman groups, including those used by TAMARIN and ProVerif. Models of Diffie Hellman groups have been the subject of considerable research over the past two decades. However, for brevity we focus only on the latest models used by these tools. Similarly, we focus on what is captured by these models, rather than how these models are made tractable behind the scenes.

We first note a shortcoming of all automated symbolic DH models: they only model exponentiation in the Diffie Hellman group, and do not capture direct use of the group operation. This is due to technical limitations imposed by the underlying logical framework which is founded on unification theory. Modelling both the group operation and exponentiation directly would require solving unification problems in a field, which is known to be undecidable [42, p. 160]. This means that these tools can model protocols which make use of exponentiation, but not those which also use the group operation directly. For example the term $(g^x)^y = g^{xy}$ can be represented, but $g^x \cdot g^y = g^{x+y}$ cannot.

Fortunately, relatively few security protocols make direct use of the group operation. This problem was considered in a recent paper: Liskov and Thayer [35] investigated the connections between computational models of Diffie Hellman groups and their symbolic counterparts. Perhaps surprisingly, they were able to show that derivability in the restricted model without direct access to the group operation, coincides with derivability in the full model with group operations. This is a deeply encouraging result, as it suggests existing symbolic tools are not missing attacks by omitting the group operation. Even more interestingly, the researchers claim a computational soundness result for prime order groups in the restricted Diffie Hellman model.

We now discuss in detail the models of Diffie-Hellman groups currently used by TAMARIN and ProVerif.

A. ProVerif's Model

ProVerif includes a standard DH model, which has been extended in two different directions. Unfortunately, neither extension is compatible with the other, so we describe all three here.

1) Standard Model: This model is included in the main ProVerif tool and is the most commonly used [13, Page 41].

```
equations: g/0, exp/2
functions: exp(exp(g,x),y) = exp(exp(g,y),x)
```

The first line introduces a constant g and a binary function exp. The second line specifies the commutativity of exp. By default, ProVerif uses a coarse model of Diffie Hellman groups, modelling the bare minimum required to allow protocols to function, and considerably restricts the attacker's power.

There is no representation of exp's associativity, inverses, or the identity element. This can lead to missed attacks. For example, if the adversary learns the term g^{xyz} and x, the adversary cannot deduce g^{yz}. Similarly, commutativity only applies to the base term. E.g:

$$g^{xyz} = g^{yxz} \neq g^{zxy} = g^{xzy}$$

2) Küsters and Truderung's Extension: In 2009, Küsters and Truderung developed a technique to allow ProVerif to handle a more granular model of DH groups [31]. They provide a pre-processor that can take a protocol description with the granular model and transforms it into a syntactic theory where these equations do not need to be considered, which can directly then be analysed by ProVerif. They consider the equations:

```
equations: (-1)/1, exp/2
functions: exp(exp(x,y),z) = exp(exp(x,z),y)
           exp(exp(x,y),y^{-1}) = x
           {(x^{-1})}^{-1} = x
```

Thus exponentiation is commutative, associative and has inverses, although not the identity element. Unlike TAMARIN's approach, this approach only works for Horn clauses with ground exponents, which means that they cannot find certain attacks where the adversary sends products. The transformation additionally causes an increase in complexity of the model, and to the best of our knowledge, this approach has not been used in modern ProVerif analysis of protocols that use DH.

3) Bhargavan et al's Extension: In 2015, Bhargavan, Delignat-Lavaud, and Pironti presented [8] the following model, built on top of ProVerif's standard model. Their intent is to approximate the presence of a "bad group" where small order elements exist.

```
functions: exp/2 gEl/2 gGrp/1 bGrp/0 bEl/1
equations: exp(gEl(gGrp(id),x),y) = gEl(gGrp(id),exp(x,
     y))
           exp(gEl(bGrp,x),y) =bEl(bGrp)
           exp(bEl(gr),y) =bEl(gr)
```

The first equation defines normal exponentiation. The second and third collapse the result of a "bad" exponentiation (either in a bad group or a bad element in a good group) to a single element. This over approximates the presence of a small subgroup, in which every operation collapses to a single element. In their paper, Bhargavan et al use this model for attack finding and automatically discovered unknown key share attacks on real world protocols. However, the model is not suitable for verification or more nuanced analysis, as a considerable range of valid attacker strategies and protocol mitigations cannot be represented.

B. TAMARIN*'s Model*

In 2012, Schmidt et al [43] presented the TAMARIN prover for the symbolic analysis of security protocols. TAMARIN supports "best in class" DH theories with the following model:

```
functions: */2 ^/2 (-1)/1 (1)/0
equations:
       x * (y * z) = (x * y) * z
       x * y = y * x
       x * 1 = x
       x * x^{-1} = 1

       {x^{-1}}^-1 = x
       (x^y)^z = x ^ (y * z)
       x^1 = x
```

The first four equations specify the group structure of multiplication in exponent, whilst the latter three describe the relationship between exponentiation of group elements and multiplication of exponents. This accurately models the multiplication in the exponent, including associativity, commutativity, inverses and identity. However, as previously mentioned, it does not allow for addition in the exponent, or equivalently the direct application of a group operation.

C. Summary

These models offer a well justified representation of a Diffie Hellman group, if we assume that the protocol uses a prime order group, correctly rejects the identity element and can unambiguously parse group elements. In the real world, protocols often work in non-prime order groups, accept the identity element and must take great care when handling untrusted 'group elements'. In the following section, we make our first contribution and investigate how these models can be extended to handle the non-prime order case.

IV. NEW SYMBOLIC MODELS FOR DH GROUPS

We now develop our first main contribution: more accurate symbolic models of non-prime order groups. We start out by describing several behaviours of such groups, then introduce a more fine grained element representation, after which we present three models in detail in sections IV-C to IV-E.

A. Behaviours of non-prime order groups

In the prime order case, by Lagrange's Theorem (§II-A), every element, other than the identity element, must have prime order and hence there cannot be any small subgroups. However, this is not necessarily true in the non-prime order case. A non-identity element may have an order lower than the order of the group and instead of generating the entire group, the element will generate a subgroup. For example, an element h with order 2 generates a subgroup of size 2, with $h^2 = id$, $h^3 = h$ and so on. Furthermore, *every* non-prime order group necessarily contains additional subgroups, due to Cauchy's Theorem (§II-A). We call elements which belong to at least one small subgroup, *small subgroup elements*. These small subgroup elements, allow for a number of additional behaviours which aren't found in prime order subgroups.

Firstly, *confinement*, when a small order element is exponentiated, the possible results are limited to other elements in the subgroup. Assuming the exponent was randomly sampled, this leads to a non-negligible likelihood of collisions where ${h_1}^x = {h_2}^y$ for h_i small order elements and x, y randomly sampled integers. In contrast, if the h_i are drawn from the prime order subgroup, collisions only happen with negligible probability.

This can seriously impact the security of higher level protocols which rely on the 'contributivity' of Diffie Hellman operations as we will see in §VII.

The second type of behaviour is that of *point mangling*, where given an element g^x in the prime order subgroup and h an element of low order, we combine them to get $h \cdot g^x$ an element in the supergroup. Although $h \cdot g^x$ is a distinct bit string from g^x, with non-negligible probability we may have $(h \cdot g^x)^z = g^{xz}$. This can impact protocols implementing replay detection or other countermeasures, as they may expect public keys to be unique.

Finally, it is possible to exploit small order subgroups to perform *key leakage attacks*. In 1997, Lim and Lee [34] introduced this attack, permitting the recovery of secret exponents used in Diffie Hellman operations. The crux of the technique is that the attacker submits an element outside the prime order subgroup to a protocol, which then uses the element in an exponentiation with a secret exponent. The attacker then observes the resulting behaviour of the protocol. If the attacker can deduce or confirm a guess of what the result of the exponentiation was, they can learn a limited amount of information about the secret exponent which was used. The attacker can then repeat this process and eventually combine the information in each guess using the Chinese Remainder Theorem and recover the secret exponent in its entirety. This attack has proven practical in many real world situations [36, 37, 54] including recent catastrophic key recovery attacks which impacted OpenSSL, the Exim mail server, as well as many other TLS, SSH and IKE implementations. With these behaviours in mind, we now set out to build symbolic models which can capture these behaviours.

B. Element Representation

Our first question is how to represent group elements in a more complex group. We note that because the number of operations to solve the discrete log problem in a group scales with the size of the largest prime factor of the group order, all Diffie Hellman groups must have at least one large prime factor. Using the fundamental theorem of finitely generated abelian groups (§II-A), which states that for any finite abelian group G of order n with a non-repeated large prime factor p, there is an isomorphism such that $G \cong H_1 \times H_2$ where $H_2 \cong \mathbb{Z}_p$ and H_1 is of order h such that $n = hp$. This means that any element in G can be expressed in the form:

$$h \cdot g^y \cong (h, g^y)$$

where on the right hand side we have (partially) decomposed the group. We will informally refer to H_1 as the cogroup of H_2. In the (cryptographically unusual) case that p is a repeated factor, the prime order subgroup may be contained in $\mathbb{Z}_{p^k}$ for some k, but this does not otherwise change our discussion.

We sketch the intuition behind this idea, noting that the underlying mathematics is well known: any finite abelian group has a finite set of generators. Any element can be written as a combination of some powers of these generators. This isomorphism is just between the 'natural' representation of the group, and the representation of the group in terms of an integer associated to each generator. Instead of performing the group operation on elements of the group, we may equivalently add the powers of each generator. Similarly, exponentiation distributes over each element in the tuple in the natural way. This allows us to describe any element of a finite abelian group in one the following four forms:

- (id, id) is the identity element.
- (id, g^x) is a regular element of the prime order subgroup.
- (h, id) is an element in the cogroup.
- (h, g^x) is an element in the supergroup.

We will now use this decomposition to provide a general representation of group elements in the symbolic model. We define the function symbol $ele(t, s, n)$ with arity 3. The first argument t represents the identifier of the group, which will be useful when we consider protocols using multiple distinct groups later. s represents the cogroup component, n is the prime order component. We will elaborate on each of these components in the following three sections.

In the following, we differentiate between three different types of group. Firstly, in §IV-C we consider prime subgroups and make a modest extension to adapt traditional models to our new representation. Then in Section IV-D we consider 'nearly-prime' groups where the order of the group n can be written as hp with p a large prime and h a small cofactor. Finally, In Section IV-E we then deal with the general case, where h may even be much larger than p.

C. Modelling Prime Order Groups

We now present a small extension to TAMARIN's default model, which captures the identity element in a prime order group. Note, this is the identity element for the group operation, rather than multiplication in the exponent. The ProVerif extension from [8] provides a model suitable for the identity element but uses less accurate DH equations than TAMARIN.

We can get the best of both worlds by by extending TAMARIN's equational theory for Diffie Hellman groups defined in §III-B with an additional constant gid and equation:

$$gid^x = gid$$

We combine this model with our new representation, and model a prime order group G as elements of the form:

$$ele('G', gid, X)$$

Here, $'G'$ is a public constant representing the name of the group, s, the cogroup component is fixed as the identity element. Finally, X will be a regular Diffie Hellman term using the exp operator discussed earlier.

As we are working in the symbolic model, we must explicitly allow the adversary to extract information from elements. The first two parameters are public, so we need only allow the adversary to extract the third term:

```
functions: ele/3 extract/1
equations: extract_1(ele(t,s,n)) = n
```

Introducing this equation allows for non-contributive behaviour that was previously not captured in TAMARIN under the implicit assumption that protocols rejected the identity element as a valid point. Later, we will see how to restore the identity element check in situations where a protocol explicitly requests it. This model is the starting point for our work.

D. Modelling "Nearly-Prime" Order Groups

With this representation in mind, we now consider the case of 'nearly-prime' groups, before tackling the more general case in §IV-E. We consider a group to have nearly-prime order if it can be written as $n = hp$ where h is a number such that $log_2(h) \ll log_2(p)$. That is, the additional factor h is very small compared to the order of the prime subgroup. This means the cogroup is of small size, leading to the confinement and point mangling behaviours from Section IV-A. However, due to its small size, the leakage attack of Lim and Lee is not of practical impact, leaking only a limited number of bits and we will not consider it for now.

The question arises of how to model operations in the small order cogroup. Depending on the group in question this cogroup may contain further small order groups or have other internal structure. To provide a general model, we will abstract these details away by allowing the adversary to choose how this small subgroup operates. We justify this approach on two grounds. Firstly, we intend to give the adversary as much as is reasonable to ensure our verification results are meaningful. Secondly, protocols are typically considered secure if they can be implemented with a wide class of groups, consequently, they do not and indeed cannot rely on the exact equations which hold in a specific small order subgroup. There also exist various mitigation strategies that we will discuss later in §VI.

We model this behaviour in our symbolic model by providing the adversary with a representation of these small order points and we provide a private channel by which the adversary may 'program' the result of exponentiation with a small order component. This allows the adversary to force collisions or inequalities according to their requirements, and consequently the adversary will always know the small order component of a group element term.

Consider the following example rule in TAMARIN's notation:

```
rule Operate:
  [In(X),State(y)]-->[Out(X^y)]
```

We perform the following steps:

1) Replace X^y with $ele(t, r, n^y)$.
2) Replace X with $ele(t, s, n)$.
3) Add the premise: $In(r)$.
4) Add the annotation: $Raised(t, s, r, y)$.

Thus the rule becomes:

```
  rule Operate:
  [In(ele(t,s,n)),State(y),In(r)]
  --[Raised(t,s,r,y)]->
  [Out(ele(t,r,n^y))]
```

Note that in practice, we perform this transformation automatically. This provides a private channel by which the adversary, using the message fact *In*, can determine the outcome of an operation in the small subgroup. In this case specifying that $s^y = r$.

However, an unrestricted private channel is too powerful. Firstly, the adversary is not required to operate the channel deterministically, meaning that repeating the same operation may lead to different results. Secondly, if the small subgroup component is the identity element gid, we know the result of exponentiating must be the identity element as $gid^x = gid$. We use the *Raised* label and TAMARIN's restriction system to enforce these constraints:

$$\texttt{Consistency} : \forall t, s, r_1, r_2, y, \#i, \#j \;. \\ Raised(t, s, r_1, y)@i \wedge Raised(t, s, r_2, y)@j \\ \implies r_1 = r_2$$

$$\texttt{Identity} : \forall t, r, y, \#i \;. \\ Raised(t, gid, r, y)@i \Rightarrow r = gid$$

We now explore the behaviour of this model: When the input is the identity: $ele('G', gid, gid)$, the result is necessarily also the identity. Similarly a prime order element $ele('G', gid, X)$ is also well behaved and will produce $ele('G', gid, X^y)$. Small subgroup elements are controlled by the attacker and consequently $ele('G', s, gid)$ will become $ele('G', z, gid)$ with z selected by the adversary, allowing for confinement and collisions. Finally supergroup elements $ele('G', s, X)$ are partially controlled by the adversary, as X^y is determined using TAMARIN's prime order model, but control of the s term allows for point mangling.

E. Modelling Composite Groups

We now turn to more general groups, where we do not require any relationship between the prime order subgroup and the size of the supergroup. Whilst these groups are not typically considered desirable to work in from a security perspective, they have often been selected for reasons of performance. In addition to the behaviour found in 'nearly-prime' groups, these groups allow for the key leakage attack of Lim and Lee to be performed.

We build on our earlier model of nearly-prime groups. As we have already allowed the small order group in that case to grow to arbitrary size and the adversary to determine all operations in the small order group, the larger cogroup in this case requires no additional extension beyond the modelling the information leak induced by the key leakage attack. We first consider the properties of this attack. There are a few fundamental requirements for this attack to succeed:

1) The same exponent must be reused in multiple calculations.
2) The attacker must be able to submit an element with a low order component.
3) The attacker must be able to confirm or invalidate a guess at the result. This may be an offline computation or online interaction, but even a single guess which can be confirmed or rejected suffices.
4) The supergroup must have enough small order factors to allow for the recovery of a significant fraction of the key.

Note that in the 'Nearly-Prime' setting, the low cofactor ensures this last condition is not satisfied. Three of these properties (1,2,4) are easily represented as trace properties as they are predicated on a particular sequence of events. However, the third condition is not a trace property but a hyper property as it requires reasoning about alternative possibilities. Whilst hyper properties can be explored in the symbolic model, support is considerably more limited than for trace properties.

Consequently, we first describe a trace property which captures conditions 1, 2, and 4. If this trace property holds, we can be sure that no small subgroup key leakage attack is possible on the protocol:

$$\neg\exists t_1, l_1, r_1, t_2, l_2, r_2, y, \#i, \#j. \\ Raised(t_1, l_1, r_1, y)@i \wedge Raised(t_2, l_2, r_2, y)@j \wedge \\ r_1 \neq r_2 \wedge l_1 \neq gid \wedge l_2 \neq gid$$

If this property holds, then there is no trace satisfying all of conditions 1,2, and 4. The presence of two raised labels with the same exponent y ensures condition 1. Likewise, the final two clauses ensure condition two holds. Condition 4 is known in advance by selection of the group, if the protocol uses more than one group and only a subset of these groups are composite, then this lemma can be specialised, by further requiring that t_1, t_2 be equal to the types of the composite group. We discuss this further in §B.

However, if this trace property does not hold, the possibility of an attack depends on condition 3. Although a real-or-random indistinguishability test on subgroup elements is appropriate for testing this condition, and can be specified in the symbolic model, tool support for indistinguishability testing is still in its infancy. Consequently, although we developed a model suitable for toy examples, it was not tractable for real world protocols.

We leave this as an area for future work. However, we note all protocol mitigations of key leakage target conditions 1, 2 and 4, as condition 3 is very hard to prevent and consequently any reported attack is likely to be a practical one.

V. Symbolically Modelling Elliptic Curve points

We now turn from our discussion of internal group structure to the representation of elliptic curve elements in the symbolic model. Whilst elliptic curves have many advantages over traditional finite field DH groups, their complex representations require more careful management.

In the finite field setting, group elements are integers in the range 1 to $p-1$ for some large prime p. Although protocols will typically intend to operate within a prime order subgroup inside this range, the group operation is well defined over the entire range of elements. However, in the elliptic curve setting, group elements consist of a pair of finite field elements, that satisfy the curve equation. This is discussed in more detail in Appendix E. For points in the finite field that do not satisfy the curve equation, the group operation is not well defined and performing operations on these invalid points can lead to catastrophic results [11, 21, 30].

In the following, we provide symbolic models for the additional behaviour that elliptic curve operations can exhibit when operating on invalid points. To help modellers in choosing the right symbolic model for a particular curve, we provide a list of common curves and their properties in Appendix A.

We first consider a special case, when the curve operation is implemented using a single coordinate ladder, before providing a general model suitable for more traditional implementations. Our models build on and extend those in the previous section to handle the additional properties of elliptic curves.

A. Single Coordinate Ladders

Whilst elliptic curve points are traditionally represented by both an x coordinate and a y coordinate, both coordinates are not necessarily needed for cryptographic purposes. Using the curve equation (see Appendix E), it is possible to uniquely identify a point using its x coordinate and a single bit. The reason for this is that the curve equation can be solved to give two possible solutions for y (corresponding to the positive and negative square roots) and the single bit can be used to select the correct solution. Consequently, using both the x and y coordinate only offers one additional bit of entropy over the x coordinate alone.

Furthermore, there exist special techniques for performing exponentiation (scalar multiplication) on curve elements that only use the x coordinate. Whilst these techniques do not support more general uses of elliptic curves, such as digital signature schemes, they have become popular for Diffie Hellman operations. These techniques are known as single coordinate ladders and were first introduced by Montgomery in 1987 [38] for a special class of curves. Later Brier and Joye [15] provided a ladder suitable for any elliptic curve. In addition to requiring less bandwidth, these ladders are also easy to implement in constant time and highly performant, leading to widespread adoption.

We now consider how these ladders behave with x coordinates which are not on the correct curve. As is discussed further in Appendix E, every elliptic curve E over a finite field is uniquely associated with another elliptic curve E', known as its quadratic twist, or twist for short. For any x coordinate in a finite field, it so happens that there exists a y value such that (x, y) either lies on the intended curve, or its twist. That is, every x coordinate can be said to lie on the curve, or its twist (or both). Furthermore, a single coordinate ladder is 'well-behaved' for both a curve and its twist. Thus for any x coordinate, the ladder will calculate the group operation either on the curve, or its twist.

This has several important consequences. Firstly, modern curves have been designed to be *twist secure* [19], where performing a discrete log on the twist is also intractable. This is intended to allow for single coordinate ladders to be used without having to ensure an x coordinate lies on the correct curve. This reduces implementation complexity and improves performance. However, older curves do not necessarily have this property. For example, performing discrete logs on the twist of the NIST prime order curve P-224 takes roughly 2^{58} operations, compared to 2^{111} operations on P-224 itself [5]. Another complication is that whilst some curves may have prime order and hence no small subgroup elements, their twists may only be nearly-prime or even composite, further complicating the use of ladders on these curves.

As the properties of the twist depend on the specific curve used, we provide a model for each class of twist. For example, the twist may be represented by a prime order group, a nearly-prime group or a composite group as described in the previous section. We will introduce an additional type of group as well, where the discrete log is easy, to cover curves which do not have twist security.

We will differentiate between elements on the main curve and elements on the twist using the first parameter of $ele(t, s, n)$. Previously, we left this value unconstrained and uninterpreted. Now, we will restrict to either being $'C'$, to indicate an element on the curve or $'T'$ to indicate an element on the twist.

We now specialise protocol rules that perform exponentiation into two constrained variants. One rule variant will handle elements on the curve, and the other will handle elements on the twist. We return our earlier example from §IV-D on page section IV-D on page 5 and transform it for the single coordinate case:

```
rule Operate_Normal:
[In(ele('C',s,n)),State(y),In(r)]
--[Raised('C',s,r,y)]->
[Out(ele('C',r,n^y))]

rule Operate_Twist:
[In(ele('T',s,n)),State(y),In(r)]
--[Raised('T',s,r,y)]->
[Out(ele('T',r,n^y))]
```

We can now further restrict each rule, depending on the properties of the curve and its twist. For example, if one should be of prime order, we will set s to be gid on all rules for that curve. The situation is similar for testing for subgroup key leakage. If the twist has composite order, but not the curve, we can specialise the key leakage trace property to require that the first parameter of Raised be equal to $'T'$, ensuring only twist attacks are captured. This duplication of rules will also come in useful later when we consider protocol level mitigations in §VI. Note that in the special case where the curve and its twist have exactly the same properties and the protocol applies its mitigations correctly to both curves, we can omit this duplication of rules.

There is one additional behaviour that may be possible on a curve's twist: the discrete log problem may not be difficult. We capture this situation by providing the adversary with an oracle they can query to take the discrete log of a twist element. We first extend every Twist variant to also produce the token $TRes(n^y, y)$ and provide the rule:

```
  rule Twist_Discrete_Log:
    [In(X^y),TRes(X^y,y)]-->[Out(y)]
```

This rule requires the adversary to have learned the full Diffie Hellman value to calculate its discrete log.

B. General Invalid Curve Points

We now consider the more general setting, where points are represented as both a x and a y value. This is often the case in protocols intended for widespread adoption (such as TLS) where there may be many interoperating implementations. Unlike in the single coordinate case, the attacker is not restricted to operating on the curve or its twist, meaning twist security has little relevance in this setting.

In 2000, Biehl, Meyer and Müller [10] were the first to investigate invalid curve point attacks. By carefully selecting points, the group operation can be forced to operate in a series of different curves, each with low order points. This allows the small subgroup key leakage attack of Lim and Lee to be performed (as described in §IV-E). In 2003, Antipa et al. [1] extended this work and calculated approximate running times and the number of interactions for this style of key leakage attack. More recently, Neves and Tibouchi [39] introduced a new technique for finding useful invalid curve points on Edwards Curves. Not only does the attack find points of low order, it can also move the computation to curves where the discrete log is easy.

In summary, there are a number of general and powerful techniques for selecting invalid curve points, which allow for the behaviours we have discussed earlier: subgroup confinement, subgroup key leakage and easy discrete logs.

We now extend our earlier model of Diffie Hellman elements to represent these special behaviours in the elliptic curve setting. We will represent the elliptic curve, according to its type, as either prime, nearly-prime or composite order. Then provide a new construction capturing invalid points.

We represent elliptic curve points directly as the combination of two elements using TAMARIN's pairing operator: $\langle x, y\rangle$ where $x, y = \mathit{ele}(t, s, n)$. We then distribute exponentiation over them in the natural way. As in the earlier twist case, we will duplicate and specialise the rules which perform exponentiation.

For the first rule, we will add the constraint that $x = y$, this represents operation on a valid curve point where x and y satisfy the curve equation. For a valid point, each coordinate can be recovered from the other with only one additional bit of information, so it is reasonable to represent them as equivalent terms. Additionally, we require the first parameter of each element to be $'C'$ to reflect the well formed nature. In the second case, we will add the constraint that $x \neq y$. In this case the point is invalid and we will allow the adversary to substitute in a point of their choice.

We transform the invalid exponentiation rule by:

1) Add a Premise $In(\langle t_x, r_x, n_x, t_y, r_y, n_y\rangle)$, representing the attacker providing a substitute point
2) Write the result as $\langle \mathit{ele}(t_x, r_x, {n_x}^z), \mathit{ele}(t_y, r_y, {n_y}^z)\rangle$
3) Add the annotation $\mathit{Raised}(\langle t_x, t_y\rangle, \langle s_x, s_y\rangle, \langle r_x, r_y\rangle, z)$
4) Add the conclusion $\mathit{IRes}(\mathit{ele}(t_x, r_x, {n_x}^z), z)$
5) Add the conclusion $\mathit{IRes}(\mathit{ele}(t_y, r_y, {n_y}^z), z)$

And finally we have the rule for taking discrete logs in on invalid elliptic curves:

```
rule Invalid_Discrete_Log:
[In(X^z),IRes(X^z),z)]-->[Out(z)]
```

This model is highly flexible and describes the behaviour of invalid points. It uses elements of the models from all previous sections. It also allows the protocol to split up x and y points according to how these distinct values might be used in practice. Note that in this case, a small subgroup key leakage attack is always potentially possible using invalid curve points, so leakage lemma should always be added. As in the twist case, we can use pattern matching to ensure regular curve elements are not the subject of false attacks by requiring the lemma to pattern match on the type of the element ($\langle t_x, t_y\rangle$).

VI. Modelling Mitigations

There are several countermeasures and mitigation strategies which can be employed by protocol designers to restrict or remove the behaviours we have discussed in this paper. Often, these countermeasures have been added to implemented protocols in an ad-hoc fashion, leading to uncertainty about their suitability and effectiveness. As well as discussing these mitigations and their corresponding model, we also consider the recommendations of various authorities as to which mitigations should be employed and under what circumstances they suffice. We begin with the more traditional checks:

A. Rejecting the Identity Element

Every group contains an identity element, which is a fixed point under exponentiation. In some implementations this point is not expressible in the chosen coordinate system; others may choose to manually catch and reject this value before operating on it.

In our model, the identity element is represented by $\mathit{ele}(t, \mathit{gid}, \mathit{gid})$ in the single coordinate case and $\langle \mathit{ele}(t, \mathit{gid}, \mathit{gid}), \mathit{ele}(t, \mathit{gid}, \mathit{gid})\rangle$ in the general model of elliptic curves. Consequently, it suffices to check that at least one component is not gid to rule out the identity element.

So we provide the label: $NotID(P)$, where $P = ele(t, gid, gid)$ and the restriction:

$$\forall t, \#i.NotID(ele(t, s, n))@i \implies s \neq gid \vee n \neq gid$$

In the general model of elliptic curves, we generalise this axiom to handle both the x and y coordinate in the natural way. This check may be used in a protocol before or after exponentiation.

B. Excluding Low Order Points

In nearly-prime groups, there are a small number of elements which have low order. Excluding these points, by matching against a list, is intended to ensure 'contributory' behaviour when exponentiating points. For some elliptic curves, notably Curve25519 [6], this countermeasure has proven controversial; leading to a number of discussions between designers and implementers.

Daniel Bernstein, designer of Curve25519, recommends against excluding low order points, as this only needs to be done for 'exotic' protocols [4]. Similarly, Trevor Perrin, co-designer of Signal, recommends against the check as "safe protocols" should not require it [40]. On the other hand, some security researchers advocate [23, 49] for including the check and the IETF has opted to allow it [24]. This has lead to a similar split in implementation behaviour with LibSodium choosing to reject low order points [33], whilst the Go standard library accepts them [26]. We will see the impact of these decisions in a case study in §VII.

We represent these checks in our model by providing a label $NotLowOrder(P)$ where $P = ele(s, n)$ and the restriction:

$$\forall s, n, \#i.NotLowOrder(s, n)@i \implies n \neq gid$$

This check is not always practical, for example in composite groups there may be a huge number of low order points, which would these exclusion checks impractical. Similarly, this check does not ensure elements belong to the intended prime order subgroup, they may still have high order and belong to the larger supergroup. Consequently, this technique will prevent confinement, but not necessarily leakage.

C. Checking Element Order

This is a more thorough, but more computationally expensive, version of the previous check.

As discussed in Appendix C, the order of an element x is the smallest natural number k such that $x^k = gid$. When the order of a group is equal to hp, p is prime and h and p are coprime, then given an element in the group, checking it belongs to the order p subgroup can be done by ensuring $x^p = 1$. If the protocol *assumes* x belongs to the group, rather than *checking* it is a valid representative, the order check is not well defined. In prime order groups, it is common for this check to be replaced by checking that the group element is not the identity element, as this is much more efficient and all other elements have the same order.

We model this behaviour by ensuring that if an element is within the group, we correctly confine it to the prime order subgroup and otherwise do not impose a restriction. We provide the label: $OrdChk(P)$ and the restriction:

$$\forall t, s, n, \#i \ . \ OrdChk(ele(t, s, n))@i \implies s = gid$$

In the general elliptic curve setting, this becomes:

$$\begin{aligned}&\forall t_x, s_x, n_x, t_y, s_y, n_y, \#i \ .\\ &OrdChk(\langle ele(t_x, s_x, n_x), ele(t_y, s_y, n_y)\rangle)@i \implies\\ &t_x \neq t_y \vee s_x \neq s_y \vee n_x \neq n_y \vee (s_x = gid \wedge s_y = gid)\end{aligned}$$

This states that either the point is invalid, or it is an element of the prime order subgroup. Note, the identity element is allowed, as $gid^p = gid$.

D. Low Order Clearing

There are several techniques which allow the protocol or the implementation to "zero out" the low order component. Let n be the order of the supergroup, with p the order of the prime subgroup and h the cofactor such that $n = ph$. Then if p and h are coprime, for any element in $g \in G$ we have that g^h will belong to the prime order subgroup. For example, if g is a small order element, $g^h = gid$, which (also) belongs to the prime order subgroup.

This prevents information leaks, as an adversary can no longer deduce any information from the small subgroup component. However, this can exacerbate subgroup confinement problems, as all low order elements are confined to the same output value, the identity element.

This technique is known by a number of names and can be implemented in different ways. For example, in Curve25519, this is know as private key clamping and any private key is manipulated by the implementation to ensure it is a multiple of the cofactor [6]. Similarly, protocols can use this mitigation by raising elements to the power of the group cofactor before using them. This does also change the high order component of an element and consequently all implementations of a protocol must agree on whether to use this mitigation.

We model this by providing the label $ClearPoint(P)$. We substitute $P = ele(t, s, n)$ with $P' = ele(t, gid, n^{'h'})$. $'h'$ is a public constant which represents the change to the prime order component and we assume is already known to the adversary.

E. Checking the Curve Equation

When dealing with elliptic curve points, it is important to ensure they belong to the correct curve before operating on them, as discussed in §V. Even if using a single coordinate ladder, this is an important step when the curve is not twist secure.

In the single coordinate case, we provide the label $NoTwist(X)$ and the restriction:

$$\forall t, s, n, \#i \ . \ NoTwist(ele(t, s, n))@i \implies t \neq' T'$$

This rejects any point on the twist. Note that if the curve check is applied immediately prior to using the point in an exponentiation, it is equivalent (and more performant in TAMARIN) to simply delete the rule variant for the twist.

However, this check might be made at a different point in the protocol and hence we provide this rule.

In our general model of elliptic curves, we model this check by using the label $Eq(x, y)$ which enforces the equality (and in our model validity) of the curve point. If this check is performed in the same protocol rule as the exponentiation, we can simply omit the generation of the invalid point variant as it is trivially impossible.

F. Summary

These mitigations cover the typical types of mitigation available to protocol designers and recommended by standards bodies. For example, NIST's guidance [3, §5.6.2.3.3] recommends an elaborate sequence of checks and defence in depth measures when exponentiating the point Q:

- Ensure Q is not the identity element (§VI-A)
- Verify Q satisfies the curve equation. (§VI-E)
- Perform a full order check (§VI-C)
- Raising by the cofactor during exponentiation (§VI-D)
- Rejecting the result if it is the identity element (§VI-A)

Contrastingly, Curve25519 implementations will always raise elements by the cofactor, and may optionally reject low order points, but do not perform any other checks. This is in part because the single coordinate ladder and twist security allows for some checks to be omitted and in part because small subgroup confinement is not considered problematic by the designers. One additional mitigation we have not discussed here are the Decaf [28] and Ristretto [53] proposals which build an efficient abstraction of a prime order group using either Curve25519 or Curve448. These proposals are exciting as they allow for Curve25519 to be treated as a prime order group, removing the need for other mitigations, but have not yet seen widespread usage.

VII. Case Studies

Over the proceeding sections we have built a family of symbolic models for more accurate modelling of DH groups. We provide an appendix (§A) describing how to select a particular model for a given group or curve, as well as advice on how to proceed if the group or curve is unknown. We now evaluate our models on three real world case studies.

Our source code and models are available [20].

A. Scuttlebutt

Secure Scuttlebutt is a peer-to-peer gossip protocol for the distributed web [48]. Users run their own Secure Scuttlebutt endpoint locally and connect to their contact's endpoints to exchange messages and synchronise state. There are several interoperating implementations, including Javascript, Go, Rust, and C, representing the popularity of Scuttlebutt [46].

Here we focus on a specific component of Secure Scuttlebutt: the so-called *secret-handshake*, which is used to authenticate and encrypt connections between peers. This handshake is of critical importance to the higher level protocol, as application data is typically otherwise unencrypted. The Secret Handshake was described in an early whitepaper [50] and wire specification [44]. The design draws on lessons from the older Station-to-Station protocol and the more recent Noise Protocol framework and aims to ensure not only basic security requirements but also perfect forward secrecy, resistance to unknown key share attacks, and key compromise impersonation. It also aims for a less typical authorisation property: the secret handshake is intended to keep the responder's *public* key secret and only allow connections from initiators who already know the correct public key. This mechanism forms the basis of Scuttlebutt's invitation system, where peers can 'friend' each other by exchanging their public keys. The handshake was originally modelled in Tamarin in February 2018, using the traditional model of Diffie Hellman groups [47].

The secret handshake is depicted in Figure 1. i, r are the private signing keys of the initiator and responder, and g^i, g^r are the corresponding public keys. There are two distinct phases. Firstly, each party exchanges an ephemeral Curve25519 public key and proves knowledge of C, which is a constant used to identify the Scuttlebutt Network. The initiator then derives a key K_1 using the ephemeral values and the responder's public key and uses it to transmit both their own public key and an Ed25519 signature 'proving' they intend to talk to the responder. The responder can decrypt this message, learn the initiator's public key g^i and then prove control of their own public key g^r. The resulting key is then used to encrypt and authenticate further messages used in the higher level gossip protocols. An important authentication property is that *only an initiator who already knows the responder's public key, should be able to complete a handshake*. We modelled this protocol in Tamarin, using our 'nearly-prime single coordinate elliptic curve model with nearly-prime twist' model of Curve25519.

Tamarin automatically found a novel, previously unreported attack on this protocol which violates this authentication property. That is, an adversary with no knowledge of the responder's public key is able to complete a handshake as an initiator and knows the final key. This attack makes use of the small subgroup properties we have developed in this paper, and proceeds as follows. The attacker, as an initiator, selects their public key and ephemeral key to both be points of low order on Curve25519. The resulting shared key is constant and hence the adversary is able to derive the first ciphertext. However, the adversary must now produce a signature on the responder's public key, without knowing the responder's public key. Perhaps counter intuitively, this is possible and in fact, the method of producing such a signature is described in the original Ed25519 design paper [7, Page 7]. This additional signature property has recently been added to Tamarin [29].

As this attack makes use of two unusual properties of Curve25519 keys, one might assume that this attack is only theoretical. However, we successfully implemented the attack against the Go implementation of the 'Secret Handshake' [45], confirming the accuracy of our analysis. The source code of our implementation is also contained in our supplementary materials [20]. Using this attack, an adversary could compromise the privacy of every Secure Scuttlebutt user, by methodically scanning the Internet for clients running Secure Scuttlebutt

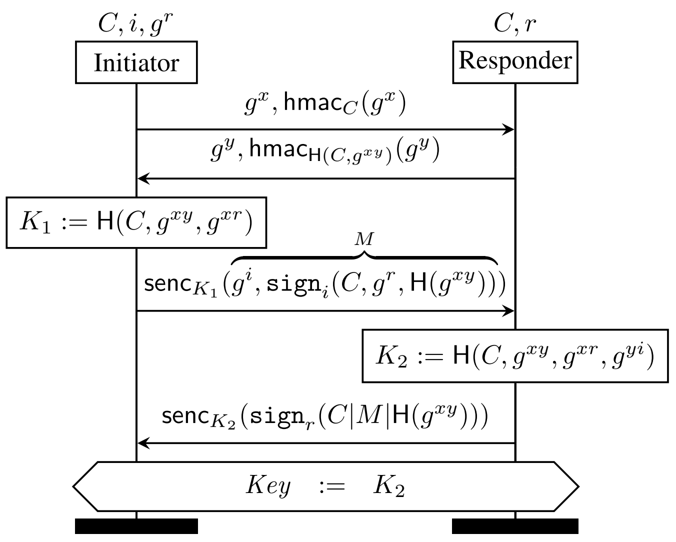
C, i, g^r
Initiator
C, r
Responder
$g^x, \mathsf{hmac}_C(g^x)$
$g^y, \mathsf{hmac}_{\mathsf{H}(C,g^{xy})}(g^y)$
$K_1 := \mathsf{H}(C, g^{xy}, g^{xr})$
$\mathsf{senc}_{K_1}(\overbrace{g^i, \mathtt{sign}_i(C, g^r, \mathsf{H}(g^{xy}))}^{M})$
$K_2 := \mathsf{H}(C, g^{xy}, g^{xr}, g^{yi})$
$\mathsf{senc}_{K_2}(\mathtt{sign}_r(C|M|\mathsf{H}(g^{xy})))$
$Key \ := \ K_2$

Figure 1: Secure Scuttlebutt's 'Secret Handshake'

and then connecting to them using our attack. These clients would then gossip their private state to the attacker.

Consequently, we have disclosed our findings to the Secure Scuttlebutt team and have recommended two different mitigations. Firstly, Secure Scuttlebutt clients could be configured to reject low order points. This ensures the attacker cannot derive the symmetric key without knowing the responder's public key and does not require a protocol change, as honest clients will never send such malformed public keys. Our alternative suggestion is to include the initiator's and responder's public key when deriving K_1 and K_2. Whilst this second suggestion requires a new version of the protocol and risks incompatibility with older clients, it ensures implementations will not accidentally (and silently) forget the low order point check. These two suggestions mirror the recommendations of protocol and primitive designers in §VI-B - it is possible to fix contributivity problems at either the primitive or the protocol level. We also provide models of our two suggested fixes, which TAMARIN automatically proves correct.

The Scuttlebutt team responded promptly and acknowledged the security issue. Interestingly, it transpired that even though all Scuttebutt implementations use the same NaCl API to handle their Curve25519 operations, not all Scuttlebutt implementations were vulnerable to our attack.

Through a joint investigation with the Scuttlebutt team, we discovered that libraries offering NaCl support have unknowingly diverged and handle low order points differently. LibSodium [32], one of the most popular libraries, rejects low order points before they are operated on. However, HACL [55] - a formally verified library advertising itself as a "drop in" replacement for LibSodium - omits this check, as does the NaCl implementation in Go's standard library [26]. Consequently, a protocol relying on contributivity, implemented using the NaCl API, may or may not be vulnerable depending on the underlying library.

Our work has lead to several changes. The Scuttlebutt team opted for our first proposed fix, as backwards compatibility is an important real world concern, and they have updated the vulnerable Scuttlebutt clients to ensure low order points are rejected. We also raised the omitted check with the maintainers of Go's NaCl API and the HACL library, and they both agreed to add this check in future releases.

B. Bluetooth Handshakes

The Bluetooth standard [27] has a long and convoluted history of overlapping standards, design flaws and vulnerable implementations. One of its core security goals is to establish a secure channel between two devices, without relying on complex configuration or prior exchange of secret information.

We focus on two handshakes defined in the Bluetooth specification: Secure Simple Pairing (SSP), introduced in 2007 and Low Energy Secure Connections (LESC), introduced in 2014. Whilst there are considerable differences in how these two handshakes behave at a low level, accounting for changes in Bluetooth packet formats and advances in hardware manufacturing, these two protocols are equivalent from a design perspective. Unlike earlier Bluetooth handshakes, these two protocols were widely considered to be secure and underwent multiple formal analyses using ProVerif [2, 16, 17].

Two devices perform a handshake by exchanging Diffie Hellman public keys on the NIST P-224 Elliptic Curve, along with some metadata. Next, each device displays a short numeric code which is intended to be a short confirmation code of the earlier exchange. The user checks that both codes match and then the two devices each prove knowledge of the shared secret key. We depict this in Figure 2. The importance of public key validation was documented in the standard and implementers of the protocol were recommended to either:

- use an ephemeral Diffie Hellman key, or
- verify the points are on the correct curve.

Perhaps unsurprisingly, implementers almost universally opted for the first mitigation, as it requires very little additional code. However, as was discovered in 2018 by Biham and Neumann [11], the first mitigation is not sufficient to secure the handshake. The issue impacted all major Bluetooth vendors, including Qualcomm, Broadcom, Intel and Google.

Biham and Neumann observed that the Bluetooth standard requires both the x and y coordinate of the Diffie Hellman key be transmitted, however only the x coordinate is included in the short numeric code that the user checks. This allows an attacker to silently change the y coordinate, leading to an invalid curve attack. Biham and Neumann perform this attack, setting the y coordinate to 0 which in turn ensures the point $(x, 0)$ will have order 2. This means each device will compute one of two possible keys, either $(x, 0)$ or $\mathcal{O}$, the point at infinity, which are both known to the attacker.

Using our improved symbolic model for elliptic curve elements, TAMARIN automatically finds this attack. We model P-224 as a prime order elliptic curve with general coordinates. We systematically investigate the proposed mitigations and claims by Biham and Neumann, as well as proposing one of our own. We also explore our model of subgroup key leakage

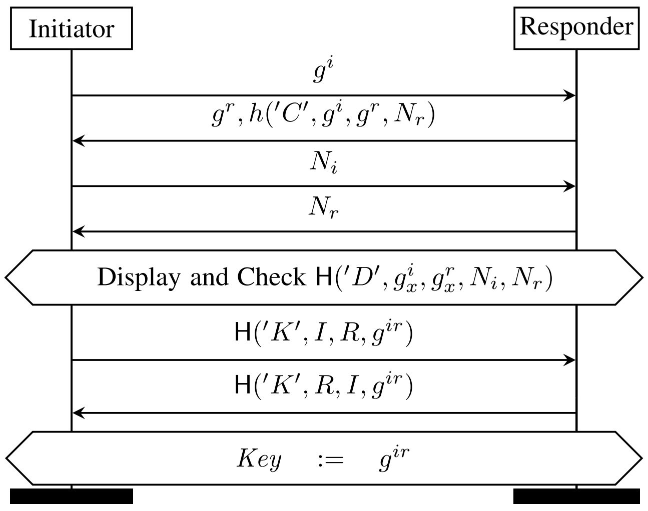
Initiator
Responder
g^i
$g^r, h('C', g^i, g^r, N_r)$
N_i
N_r
Display and Check $\mathsf{H}('D', g^i_x, g^r_x, N_i, N_r)$
$\mathsf{H}('K', I, R, g^{ir})$
$\mathsf{H}('K', R, I, g^{ir})$
$Key := g^{ir}$

Figure 2: Bluetooth's Simple Secure Pairing Handshake

and evaluate how different mitigations allow or prevent this type of attack. Our full results are documented in Table I.

Our results show, as expected, that if the implementation used static keys rather than ephemeral keys, a key leakage recovery attack could have been performed. Additionally, we propose an alternative protocol level mitigation of authenticating both the x and y coordinates and show this also mitigates the attack.

Surprisingly, we find a new attack that contradicts one of the claims of Biham and Neumann. They state in their disclosure that as long as at least one device is patched against their attack, the resulting handshake is secure. We show that is not true: it is possible for an attacker to forge the packets such that whilst the patched device will believe the handshake failed due to a network error, the unpatched device will accept the handshake and believe pairing has succeeded. Consequently, whilst an attacker may not learn confidential information exchanged over the Bluetooth link, as only one honest device is on the link, they may still use the link to compromise resources (data, sensors) on the unpatched device.

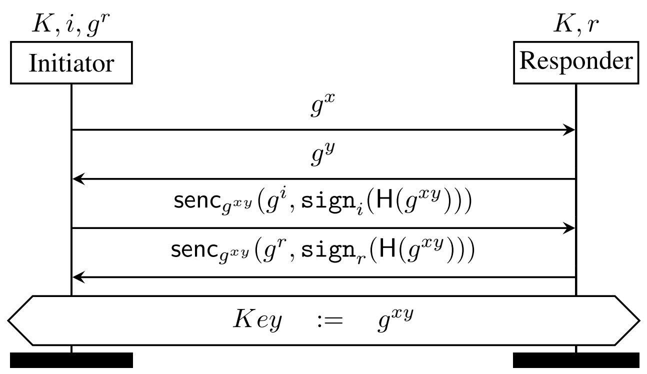
K, i, g^r
K, r
Initiator
Responder
g^x
g^y
$\mathsf{senc}_{g^{xy}}(g^i, \mathtt{sign}_i(\mathsf{H}(g^{xy})))$
$\mathsf{senc}_{g^{xy}}(g^r, \mathtt{sign}_r(\mathsf{H}(g^{xy})))$
$Key := g^{xy}$

Figure 3: Tendermint's Handshake

C. Tendermint

Tendermint [52] is a 'Blockchain API' which can be used to implement byzantine fault tolerance for arbitrary state machines. One of its features is a secure peer-to-peer layer which authenticates and encrypts connections between clients in a private network. They use a custom handshake [51] which they describe as being based on the Station-to-Station protocol. The protocol, depicted in Figure 3, performs an ephemeral Diffie Hellman handshake over Curve25519 and derives a key and a challenge value from the shared secret. Each party then signs the challenge value and transmits it to the other. Verifying these signatures over the challenge value is intended to prove both parties are using the same secure channel.

We model this protocol in TAMARIN with our enhanced DH models and evaluate its security properties. Specifically, we focus on the secrecy of the derived keys and the authenticity of completed handshakes. If an honest party concludes a handshake with another party whose key is not compromised, the resulting key should be secret from the adversary.

TAMARIN automatically discovers an attack using small subgroups, which allows an attacker to steal the identity of any party willing to complete a handshake with them. That is, if the attacker completes a handshake with A, the attacker can now connect to B using A's identity. This violates both the secrecy and the authenticity properties mentioned earlier. The attack is relatively simple, if an attacker sends a small order point as their ephemeral public key, the resulting challenge is constant, regardless of the other party's choice of key. When the handshake completes, the attacker obtains a signature on this constant challenge from the honest participant, which can then be substituted as their own signature in future sessions.

This attack was independently discovered by another security researcher and disclosed to the Tendermint team [41]. The Tendermint team proposed two possible mitigations, the first and easiest being to reject low order points and the second to include the identities of each party in the challenge derivation. We model both mitigations and verify they prevent the attack.

VIII. CONCLUSIONS

In this work, we have identified and addressed several shortcomings of traditional symbolic models of Diffie Hellman groups. We have introduced models to capture rich group structure such as small subgroups and key leakage attacks, and explored the additional attacks and capabilities of an attacker who exploits invalid elliptic curve points. We have provided a family of symbolic models, which is suitable for verification as well as attack finding, and implemented them in TAMARIN.

Through a series of case studies, we have not only evaluated the effectiveness of our models, but also found new, real world, attacks on deployed protocols and refined existing results for previously analysed protocols. In order to prove our attack is not just theoretical, we implemented our attack and verified Scuttlebutt's Go client was vulnerable. Furthermore, our work has already had real world impact with mitigations being deployed in Scuttlebutt at the application level, as well as in the NaCl API's offered in HACL and Go's standard library.

Protocol	Variant	Secure?	Leakage?	T (min)
Bluetooth P-224	Original	●	✓	5
	No Ephems	●	●	6
	NoEph + ChkC	✓	✓	<1
	Chk Curve	✓	✓	<1
	Auth both	✓	✓	<1
	Patch One	●	✓	5
Scuttlebutt Curve25519	Original	●	✓	131
	Exclude	✓	✓	88
	KDF	✓	✓	<1
Tendermint Curve25519	Original	●	✓	<1
	Exclude	✓	✓	<1
	Transcript	✓	✓	<1

Table I: Verification results when applying our various TAMARIN models to each of our case studies. Each case study terminated automatically without any user input or choice of heuristic.
Secure means the protocol achieves its stated security objectives
Leakage means the protocol is susceptible to a key recovery attack
✓ indicates that TAMARIN successfully verified the property
● indicates that TAMARIN found an attack
P-224 is modelled using a prime order elliptic curve with general coordinates; Curve25519 is modelled using a nearly-prime order elliptic curve with a single coordinate ladder and a nearly-prime order twist with low order components cleared after exponentiation.

As is typical for automated analysis performed in the symbolic model, our verification results are not proven to be computationally sound. That is, a successful verification does not necessarily imply the existence of a reduction from an attack on the protocol, to some underlying hard problem. However, we view our work as the first step towards realising a computationally sound model of real world Diffie Hellman groups. We have shown our model to be tractable and effective at capturing these new behaviours and look forward to investigating computational soundness results in future work.

With symbolic models becoming increasingly granular, automatically generating models from reference implementations looks increasingly attractive. It would be interesting to see whether recent work on automatically generating implementations can be integrated with our granular model of Diffie Hellman groups.

REFERENCES

[1] Adrian Antipa et al. 'Validation of Elliptic Curve Public Keys'. In: *Public Key Cryptography - PKC 2003, 6th International Workshop on Theory and Practice in Public Key Cryptography, Miami, FL, USA, January 6-8, 2003, Proceedings*. 2003, pp. 211–223.

[2] Kenichi Arai and Toshinobu Kaneko. 'Formal Verification of Improved Numeric Comparison Protocol for Secure Simple Paring in Bluetooth Using ProVerif'. In: *Proceedings of the International Conference on Security and Management (SAM)*. The Steering Committee of The World Congress in Computer Science, Computer . . . 2014, p. 1.

[3] Elaine Barker, Don Johnson and Miles Smid. 'NIST special publication 800-56A: Recommendation for pair-wise key establishment schemes using discrete logarithm cryptography (revised)'. In: *Comput. Secur. Natl. Inst. Stand. Technol.(NIST), Publ. by NIST* (2007).

[4] Daniel Bernstein. *Curve25519: How do I validate Curve25519 public keys?* 2006. URL: https://cr.yp.to/ecdh.html (visited on 02/02/2019).

[5] Daniel Bernstein. *Twist Security*. 2013. URL: https://safecurves.cr.yp.to/twist.html (visited on 02/02/2019).

[6] Daniel J. Bernstein. 'Curve25519: New Diffie-Hellman Speed Records'. In: *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*. 2006, pp. 207–228.

[7] Daniel J. Bernstein et al. 'High-speed high-security signatures'. In: *J. Cryptographic Engineering* 2.2 (2012), pp. 77–89.

[8] Karthikeyan Bhargavan, Antoine Delignat-Lavaud and Alfredo Pironti. 'Verified Contributive Channel Bindings for Compound Authentication'. In: *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. 2015.

[9] Karthikeyan Bhargavan et al. 'Triple Handshakes and Cookie Cutters: Breaking and Fixing Authentication over TLS'. In: *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*. 2014, pp. 98–113.

[10] Ingrid Biehl, Bernd Meyer and Volker Müller. 'Differential Fault Attacks on Elliptic Curve Cryptosystems'. In: *Advances in Cryptology - CRYPTO 2000, 20th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 2000, Proceedings*. 2000, pp. 131–146.

[11] Eli Biham and Lior Neumann. *Breaking the Bluetooth Pairing–Fixed Coordinate Invalid Curve Attack*. 2018. URL: https://www.cs.technion.ac.il/biham/BT/bt-fixed-coordinate-invalid-curve-attack.pdf.

[12] Bruno Blanchet. 'An Efficient Cryptographic Protocol Verifier Based on Prolog Rules'. In: *14th IEEE Computer Security Foundations Workshop (CSFW-14 2001), 11-13 June 2001, Cape Breton, Nova Scotia, Canada*. IEEE Computer Society, 2001, pp. 82–96.

[13] Bruno Blanchet et al. *ProVerif 2.00: Automatic Cryptographic Protocol Verifier, User Manual and Tutorial*. Version from 2018-05-16. URL: https://prosecco.gforge.inria.fr/personal/bblanche/proverif/manual.pdf.

[14] Oleg Vladimirovič Bogopolskij. *Introduction to group theory*. Vol. 6. European Mathematical Society, 2008.

[15] Eric Brier and Marc Joye. 'Weierstraß Elliptic Curves and Side-Channel Attacks'. In: *Public Key Cryptography, 5th International Workshop on Practice and Theory in Public Key Cryptosystems, PKC 2002, Paris, France, February 12-14, 2002, Proceedings*. 2002, pp. 335–345.

[16] Richard Chang and Vitaly Shmatikov. 'Formal analysis of authentication in bluetooth device pairing'. In: *FCS-ARSPA07* (2007), p. 45.

[17] Tom Chothia, Ben Smyth and Christopher Staite. 'Automatically Checking Commitment Protocols in ProVerif without False Attacks'. In: *Principles of Security and Trust - 4th International Conference, POST 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings*. 2015, pp. 137–155.

[18] *Handbook of Elliptic and Hyperelliptic Curve Cryptography*. Chapman and Hall/CRC, 2005. ISBN: 978-1-58488-518-4.

[19] Craig Costello, Patrick Longa and Michael Naehrig. 'A brief discussion on selecting new elliptic curves'. In: *Microsoft Research. Microsoft* 8 (2015).

[20] Cas Cremers and Dennis Jackson. *Supplementary Materials and Tamarin models*. 2019. URL: https://people.cispa.io/cas.cremers/tools/tamarin/primeorder/index.html (visited on 02/02/2019).

[21] *Critical vulnerability in JSON Web Encryption (JWE)*. 2017. URL: http://blog.intothesymmetry.com/2017/03/critical-vulnerability-in-json-web.html (visited on 02/02/2019).

[22] W. Diffie and M. Hellman. 'New Directions in Cryptography'. In: *IEEE Trans. Inf. Theor.* 22.6 (Sept. 2006), pp. 644–654.

[23] Thai Duon. *Why not validate Curve25519 public keys could be harmful*. 2015. URL: https://vnhacker.blogspot.com/2015/09/why-not-validating-curve25519-public.html (visited on 02/02/2019).

[24] *Elliptic Curves for Security*. 2016. URL: https://tools.ietf.org/html/rfc7748 (visited on 02/02/2019).

[25] Pierre-Alain Fouque et al. 'Fault attack on elliptic curve Montgomery ladder implementation'. In: *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*. IEEE. 2008, pp. 92–98.

[26] *Go Documentation for the Standard Library: x/crypto/nacl*. 2019. URL: https://godoc.org/golang.org/x/crypto/nacl (visited on 02/02/2019).

[27] Bluetooth Special Interest Group. *Bluetooth Core Specification v5.1*. 2019. URL: https://www.bluetooth.com/specifications/bluetooth-core-specification (visited on 02/02/2019).

[28] Mike Hamburg. 'Decaf: Eliminating Cofactors Through Point Compression'. In: *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part I*. 2015, pp. 705–723.

[29] Dennis Jackson et al. ‘Seems Legit: Automated Analysis of Subtle Attacks on Protocols that use Signatures’. In: *ACM CCS 2019: Proceedings of the 26th ACM Conference on Computer and Communications Security, London, UK*. 2019.

[30] Tibor Jager, Jörg Schwenk and Juraj Somorovsky. ‘Practical Invalid Curve Attacks on TLS-ECDH’. In: *Computer Security - ESORICS 2015 - 20th European Symposium on Research in Computer Security, Vienna, Austria, September 21-25, 2015, Proceedings, Part I*. 2015, pp. 407–425.

[31] Ralf Küsters and Tomasz Truderung. ‘Using ProVerif to Analyze Protocols with Diffie-Hellman Exponentiation’. In: *Proceedings of the 22nd IEEE Computer Security Foundations Symposium, CSF 2009, Port Jefferson, New York, USA, July 8-10, 2009*. 2009, pp. 157–171.

[32] *LibSodium: a modern, easy-to-use cryptographic library*. 2019. URL: https://download.libsodium.org/doc/ (visited on 02/02/2019).

[33] *LibSodium: reject low order points before the multiplication*. 2017. URL: https://github.com/jedisct1/libsodium/commit/c190574cee382ace1ee0f8fdacc136f1797287e6 (visited on 02/02/2019).

[34] Chae Hoon Lim and Pil Joong Lee. ‘A Key Recovery Attack on Discrete Log-based Schemes Using a Prime Order Subgroupp’. In: *Advances in Cryptology - CRYPTO ’97, 17th Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 1997, Proceedings*. 1997, pp. 249–263.

[35] Moses D Liskov and F Javier Thayer. *Formal modeling of Diffie-Hellman derivability for exploratory automated analysis*. Tech. rep. MITRE Corp Bedford MA Center for Integrated Intelligence Systems, 2013.

[36] Alfred Menezes. ‘Another look at HMQV’. In: *J. Mathematical Cryptology* 1.1 (2007), pp. 47–64.

[37] Alfred Menezes and Berkant Ustaoglu. ‘On reusing ephemeral keys in Diffie-Hellman key agreement protocols’. In: *IJACT* 2.2 (2010), pp. 154–158.

[38] Peter L Montgomery. ‘Speeding the Pollard and elliptic curve methods of factorization’. In: *Mathematics of computation* 48.177 (1987), pp. 243–264.

[39] Samuel Neves and Mehdi Tibouchi. ‘Degenerate Curve Attacks - Extending Invalid Curve Attacks to Edwards Curves and Other Models’. In: *Public-Key Cryptography - PKC 2016 - 19th IACR International Conference on Practice and Theory in Public-Key Cryptography, Taipei, Taiwan, March 6-9, 2016, Proceedings, Part II*. 2016, pp. 19–35.

[40] Trevor Perrin. *X25519 and zero outputs*. 2017. URL: https://curves.moderncrypto.narkive.com/rSFu5TRe/x25519-and-zero-outputs (visited on 02/02/2019).

[41] Harry Roberts. *Tendermint Secrect Connection Disclosure*. 2018. URL: https://github.com/tendermint/tendermint/issues/3010 (visited on 02/02/2019).

[42] Benedikt Schmidt. *Formal analysis of key exchange protocols and physical protocols*. Doctoral thesis, ETH Zurich, Switzerland. 2012.

[43] Benedikt Schmidt et al. ‘Automated Analysis of Diffie-Hellman Protocols and Advanced Security Properties’. In: *25th IEEE Computer Security Foundations Symposium, CSF 2012, Cambridge, MA, USA, June 25-27, 2012*. IEEE Computer Society, 2012, pp. 78–94.

[44] *Scuttlebutt Protocol Guide*. 2017. URL: https://ssbc.github.io/scuttlebutt-protocol-guide/ (visited on 02/02/2019).

[45] *Scuttlebutt Secret Handshake Go Implementation*. 2019. URL: https://github.com/cryptoscope/secretstream (visited on 02/02/2019).

[46] *Scuttlebutt Secret Handshake Implementations*. 2019. URL: https://github.com/auditdrivencrypto/secret-handshake (visited on 02/02/2019).

[47] *Scuttlebutt Secret Handshake in Tamarin*. 2018. URL: https://github.com/keks/tamarin-shs (visited on 02/02/2019).

[48] *Secure Scuttlebutt*. 2019. URL: https://scuttlebot.io/ (visited on 02/02/2019).

[49] Kudelski Security. *Should Curve25519 keys be validated?* 2017. URL: https://research.kudelskisecurity.com/2017/04/25/should-ecdh-keys-be-validated/ (visited on 02/02/2019).

[50] Dominic Tarr. *Designing a Secret Handshake: Authenticated Key Exchange as a Capability System*. 2015. URL: https://dominictarr.github.io/secret-handshake-paper/shs.pdf.

[51] *Tendermint Secrect Connection Protocol*. 2019. URL: https://tendermint.com/docs/tendermint-core/secure-p2p.html (visited on 02/02/2019).

[52] *Tendermint Website*. 2019. URL: https://tendermint.com/ (visited on 02/02/2019).

[53] *The Ristretto Group*. 2019. URL: https://ristretto.group/ristretto.html (visited on 02/02/2019).

[54] Luke Valenta et al. ‘Measuring small subgroup attacks against Diffie-Hellman’. In: *24th Annual Network and Distributed System Security Symposium, NDSS 2017, San Diego, California, USA, February 26 - March 1, 2017*. 2017.

[55] Jean-Karim Zinzindohoué et al. ‘HACL*: A verified modern cryptographic library’. In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2017, pp. 1789–1806.

APPENDIX

A. *Choosing a Symbolic Model*

In this section, we clarify how a particular model should be selected and employed. Our extensions exist along two axes:

- Internal Group Structure
  - Prime Order Groups (§IV-C)
  - Nearly-Prime Order Groups (§IV-D)
  - Composite Groups (§IV-E)
- Group Elements
  - Finite Field Elements (no additional behaviour)
  - Elliptic Curve Elements
    - Single Coordinate Ladders (§V-A)
    - General Coordinates (§V-B)

These models do not stand in isolation, each subsequent model extends the previous model to allow for more behaviour in a particular setting. Furthermore, we allow a wide range of mitigations to be employed by the protocol, which are described in §VI. Our models also support the use of multiple groups with distinct properties, we discuss this in §B. We now explore how to select the appropriate symbolic model for a particular real world group.

Firstly, in the finite field setting, the only relevant parameter is the group order. If it is a safe prime, e.g. $p = 2q + 1$ with q and p prime, then the group order is $2q$ and there is only a single subgroup of order 2. Consequently, the nearly-prime model is appropriate. If a DSA or Schnorr group is used, the order will have many small factors and the composite model should be used. If unsure, selecting the composite model is the most general and hence appropriate. Note there are no finite field groups with prime order, as p is prime, $p-1$ is necessarily even and hence there is always at least the subgroup of order 2.

In the elliptic curve section, the picture is more complicated. The curve itself may have prime or nearly-prime order [1]. Then it depends on the coordinate system in use. In the case of a single coordinate ladder, the twist will necessarily be either prime, nearly-prime or composite. Additionally the twist may or may not be twist secure. Alternatively in the general coordinate system, the parameters of the twist do not matter. If the exact elliptic curve is unknown, the most general model is a nearly-prime curve with general coordinates. We provide a table of common elliptic curves and their twist’s parameters in Table II. Note that Curve25519 is more tightly specified than other curves, it always applies private key clamping to eliminate low order components and is always implemented using a single coordinate ladder.

[1] Composite order curves are possible but not used in practice

Curve	Order	Twist Order	Twist Secure
NIST P-224	Prime	Composite	×
NIST P-256	Prime	Nearly-Prime	✓
NIST P-384	Prime	Prime	✓
Curve25519	Nearly-Prime	Nearly-Prime	✓
Ed448	Nearly-Prime	Nearly-Prime	✓
secp256k1	Prime	Nearly-Prime	✓
BN(2,254)	Prime	Composite	×
brainpoolP256t1	Prime	Composite	×
ANSSI FRP256v1	Prime	Composite	×

Table II: Common Elliptic Curves and their properties [5, 25]

B. Modelling Multiple Groups

We now briefly explain how to model combinations of different groups with different orders in one protocol. When a protocol uses multiple groups, it may behave differently when it receives an element of a particular group. For example, it may apply particular mitigation strategies or run certain sub protocols.

Elements of two different groups are distinguished in our framework by the first parameter. For example, $ele('G1', s_1, n_1)$ represents an element of G_1. A protocol can restrict a particular operation by pattern matching on this first parameter. Similarly, this allows us to capture the properties of different groups. Branching in TAMARIN is typically represented by having multiple rules, pattern matching or applying conditions on each rule to represent the specific conditions for that branch.

Firstly, prime order group elements should always pattern match the second parameter to gid, ensuring no small subgroup behaviour can apply. Secondly, when considering a composite group and a non-composite group, the trace based key leakage property can be restricted to require t_1, t_2 to match a particular group identity.

Elliptic curve groups can be managed in the same fashion, by following the transformations for individual rules given in each section, it is possible to model protocols where some entities use differing coordinate systems.

C. Background on Diffie Hellman Groups

We introduce the mathematical background of Diffie Hellman groups and the concepts we use in this paper. We begin with some concepts from elementary group theory, an reader without previous exposure may find [14] useful.

Definition. *Let G be a non-empty, finite, set. Let $\cdot$ denote a binary operation on G, let $\_^{-1}$ be a unary operation and $1 \in G$. We say $(G, \cdot, \_^{-1}, 1)$ is a* ***finite abelian group*** *if it has the following properties:*

1) *Closure* $\forall a, b \in G \,.\, a \cdot b \in G$
2) *Commutativity* $\forall a, b \in G \,.\, a \cdot b = b \cdot a$
3) *Associativity* $\forall a, b, c \in G \,.\, (a \cdot b) \cdot c = a \cdot (b \cdot c)$
4) *Identity* $\exists 1 \in G \,.\, \forall a \in G \,.\, a \cdot 1 = a$
5) *Invertibility* $\forall a \in G \,.\, \exists b \in G \,.\, a \cdot b = 1, a^{-1} = b$

For brevity, we will typically omit additional parameters and simply refer to a G as a group when the context clear.

Definition. *We define* ***exponentiation*** *to on a group element a by an integer x to be the result of repeated applications of the group operation. That is:*

$$a^x = \overbrace{a \cdot a \cdot \ldots \cdot a}^{|x| \text{ times}}$$

If x is negative, we replace a with a^{-1}.

The quintessential feature of a Diffie Hellman group is that performing the inverse of exponentiation is computationally intractable, i.e. given g^x, it is hard to recover x. This is known as the discrete logarithm problem.

Definition. *Let $(G, \cdot, \_^{-1}, 1)$ be a group. Let H be a subset of G. If $(H, \cdot, \_^{-1}, 1)$ forms a group, we say that H is a* ***subgroup*** *of G and respectively G is a* ***supergroup*** *of H.*

Definition. *Let G be a group and S be a subset of G. We say S* ***generates*** *G if any element in G can be expressed as a (finite) combination of elements S under application of $\cdot, -^{-1}$. Elements in S are called* ***generators*** *of G. If G can be generated by a single element, we say G is* ***cyclic****. For an element g of G, let n be the smallest natural number such that $g^n = 1$ where 1 is the identity element of G, we say the* ***order*** *of g is n and write $|g| = n$. Similarly, we say the order of G is the number of elements in the set. Note that an element of order k generates a subgroup of order k.*

Definition. *Let $(H, \cdot)$ and $(G, +)$ be two groups, a* ***group isomorphism*** *from G to H is a bijective function $f : G \to H$ such that for all $u, v \in G$ we have that $f(u+v) = f(u) * f(v)$. We write $G \cong H$.*

An isomorphism is a structure preserving map between groups, notably it preserves the algebraic properties of each group, however it does not necessarily preserve the computational properties; an isomorphism need not be effectively computable.

Note that if $|G|$ is prime, then an element is necessarily either a generator or the identity element and hence G is cyclic. Furthermore, if $|G| = n$ is cyclic then $(G, \cdot, \_^{-1}, 1)$ is isomorphic to $\mathbb{Z}/n\mathbb{Z}$ the integers modulo n.

Definition. *Let $(G, +)$ and $(H, \cdot)$ be a group, their* ***direct product*** *$G \times H$ is defined as:*

1) *The set of elements is the Cartesian product of the underlying sets. Elements are ordered pairs (g, h).*
2) *The binary operation on $G \times H$ is defined component wise: $(g_1, h_1) \cdot (g_2, h_2) = (g_1 + g_2, h_1 \cdot h_2)$*

D. Background on Finite Fields

Finite Fields were the original family of groups which were used for Diffie Hellman schemes [22]. Let p be a prime number, then the multiplicative group of the finite field $(\mathbb{Z}/p\mathbb{Z})^*$ consists of the set of the non-zero elements, i.e. the integers (technically, equivalence classes of integers $mod\ p$) between 1 and $p-1$

and the group operation is multiplication $mod\ p$. This group has order $p-1$ and is hence even. From the Fundamental Theorem of Finite Abelian Groups (FTFAG), we know that the factors of $p-1$ are crucial in understanding the internal structure of the group. There have been two common choices. The first and most popular is that of picking p such that it is a safe prime, with $p = 2q+1$ and q a prime number. This means $p-1 = 2q$ and hence necessarily there are only two subgroups: one of order 2 and one of order q. The second choice is to use a DSA group where $p = rq+1$ for some large r, often much larger than q. Here the group structure is much more complex with many small subgroups.

Group elements have a very simple representation as integers in the set $1, \ldots, p-1$ and the group operation is defined over this entire range. However, typically a protocol intends to operate in the prime order subgroup and it is only possible to tell if a group element is also a member of this prime order subgroup with careful checks.

E. Background on Elliptic Curves

Using Elliptic Curves for Diffie Hellman groups began to grow in popularity over the past 10 years and have recently become the most popular family of Diffie Hellman groups. Although they are significantly more complex to implement

Pairs of elements of K^2, written (x, y) which satisfy the above equation are considered to be points on $E(K)$, along with the distinguished identity element. Elliptic curve addition formulae are used to implement the group operation and typically operate on both the x and y coordinate. However, single coordinate ladders also exist [15, 38], which given an x coordinate and integer n can calculate x^n without needing the y component.

than finite field groups, they offer much improved performance, as well as much smaller key sizes. Elliptic curves are defined by a curve equation and a finite field of particular size. Group elements are given by the following definition:

Definition. *[18, p. 13.1] Let K be a field and let $a_1, a_2, a_3, a_4, a_6 \in K$ be elements such that the discriminant of the polynomial given in the equation below is not zero. Then the set of points $E(K)$ on the elliptic curve $E = (a_1, a_2, a_3, a_4, a_6)$ are those which satisfy the equation:*

$$\left\{(x, y) \in K^2 : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6\right\} \\ \cup \{\mathcal{O}\}$$

For this work, we do not need the full details of the group operation, or how these parameter values are selected in this paper. It is possible to choose the order of an elliptic curve group with much more control than in the finite field case. Typically, the group order is chosen to be prime or nearly-prime.

Every elliptic curve E over a field K has an associated quadratic *twist* E', which is another elliptic curve such that E and E' are isomorphic over the algebraic closure of K [18, §13.1.5]. We do not have the space to cover the full definition, proof of existence and uniqueness and other properties of the twist, see [18] for further details. However, we do need an important property, when K is a finite field [18, p. 13.17]:

$$\forall x \in K\ .\ \exists y \in K\ .\ (x, y) \in E(K) \vee (x, y) \in E'(K)$$

That is, for any x value, there is some y value such that (x, y) is on the curve or its twist. Of course, there are many more more pairs (x', y') which are not on either the curve or its twist.