CSEP 590A – Applied Cryptography

Final project

Partial overlapping multi-key encryption of structured data

Aleksander Przybylo (Student ID: 1673047)

1 Industrial motivation

The motivation for the project comes from the real problem in the manufacturing industry of sharing engineering data between the original equipment manufacturer (OEM) acting as integrator and the part suppliers. In most cases, it is up to the OEM to partition the data and distribute portions of it so that each supplier sees only what he[1] is supposed to see based on many different criteria:

- Build responsibility – the supplier is responsible for building the part defined in the engineering data
- Need to know – the supplier needs to interface with another part that is built by another supplier
- License – some manufacturing technologies and materials can be subject to export control (this is especially a concern on products under military contracts) and require the supplier to possess appropriate licenses to see.

Any method that partitions the engineering data must be 100% accurate because the OEM is liable for large fines or even cancellations of contracts in case of breaches. Other than in some exception cases (such as the aforementioned interfacing between parts built by two different suppliers), if the OEM sends data to the wrong supplier, he essentially exposes the supplier's intellectual property and can face lawsuits. Similarly, if he exposes data under export control to a supplier outside of the USA, he can face fines from the government or in extreme cases be barred from bidding on government contracts (those things can and have happened).

For this project, let's assume that we are dealing with a tree structure like XML and that each node in the XML tree represents a part (if it is the leaf) or a part assembly (otherwise). An assembly node contains the geometry of the part assembly but not the details on how to fabricate any part that composes it. We also define a relationship "interfacesWith" that creates a dependency between two parts. If partA interfaces with partB, then the owner of partA needs access to partB so that he can design partA in a way that fits with partB (e.g. the holes on partA need to align with holes on partB so that they can be bolted together). In a sense, the presence of the "interfacesWith" relationship, turns this tree structure into a graph structure which could alternatively be described using an RDF file. The list of suppliers that have access to a given part, will be referred to as the "access policy" of the part.

2 Industrial scenario

The following scenario shows a dummy Volvo V70R Bill of Materials (BOM) structure that breaks down the car into major subcomponents all the way down to single nuts and bolts.

[1] In this paper, the masculine "he" is used generically to refer to "the supplier" or "the OEM"

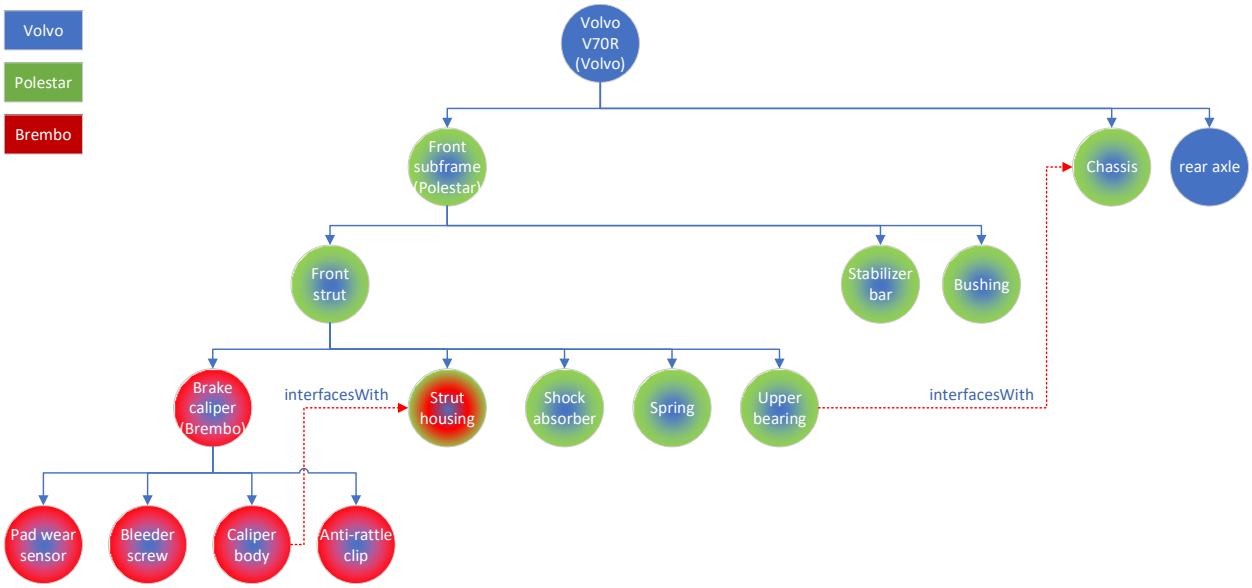

Figure 1 - Volvo V70R BOM

In the scenario on Figure 1, *Volvo* (the OEM) needs access to the entire structure (for example so that they can define maintenance manuals). Given that the "R" is a sporty model, the suspension is designed by high performance tuner *Polestar*. The braking system that composes the front suspension, is in turn designed by the Italian brake system specialist *Brembo*. Now, Brembo doesn't mind sharing their technology with *Volvo* (who is in a different business segment) but they definitely don't want to share it with *Polestar* who could easily become a competitor. The color coding shows who should have access to the part data based on the hierarchical structure and the interfacing between parts. The "Strut housing" in this scenario needs to be accessed by all three entities (*Volvo* because he acts as the integrator, *Polestar* because he is responsible for its design and *Brembo* because he needs to interface the "Caliper body" with it). The xml encoding of this tree is shown in Figure 2.

```
<?xml version="1.0" encoding="UTF-8"?>
<Car Name="Volvo V70R" Owner="Volvo">
    <Assembly Name="Front Subframe" Quantity="1" Owner="Polestar" Geometry="Front Subframe geometry">
        <Assembly Name="Front Strut" Quantity="2" Geometry="Front Strut geometry">
            <Part Name="Strut housing" Quantity="1" Geometry="Strut housing geometry"/>
            <Part Name="Upper bearing" Quantity="1" Geometry="Upper bearing geometry" interfacesWith="Chassis"/>
            <Part Name="Shock absorber" Quantity="1" Geometry="Shock absorber geometry"/>
            <Part Name="Spring" Quantity="1" Geometry="Spring geometry"/>
            <Assembly Name="Brake caliper" Quantity="1" Owner="Brembo" Geometry="Brake caliper geometry">
                <Part Name="Pad wear sensor" Quantity="1" Geometry="Pad wear sensor geometry"/>
                <Part Name="Bleeder screw" Quantity="1" Geometry="Bleeder screw geometry"/>
                <Part Name="Caliper body" Quantity="1" Geometry="Caliper body geometry" interfacesWith="Strut housing"/>
                <Part Name="Anti-rattle clip" Quantity="1" Geometry="Anti-rattle clip geometry"/>
            </Assembly>
        </Assembly>
        <Part Name="Stabilizer bar" Quantity="1" Geometry="Stabilizer bar geometry"/>
        <Part Name="Bushing" Quantity="4" Geometry="Bushing geometry"/>
    </Assembly>
    <Part Name="Chassis" Quantity="1" Geometry="Chassis geometry"/>
    <Part Name="Rear Subframe" Quantity="1" Geometry="Rear Subframe geometry"/>
</Car>
```

Figure 2 - XML encoding of the Volvo V70R BOM

The portion of the file that will actually be encoded will be the Geometry property on each Part tag. In this toy scenario, each property value is simply the Part Name plus the word "geometry". In a real-life

scenario, the value of this property would be the binary CAD data (such as CATIA, NX or Creo). The rest of the file carries no proprietary information that needs to be protected. Everyone knows that a car has a front and a real axle and one only needs to look at the brake caliper of a Volvo V70R to see "Brembo" written in big letters on it. This might not be the case in some military contracts where the supplier relationships can indeed be secret. In that case, the supplier information could simply be encrypted the same way as the Geometry property.

The following list details some high-level requirements for a valid solution:

I. The solution needs to support data distribution to all suppliers using a single encrypted file.
II. The size of the file can be larger than the unencrypted file but needs to be independent of the number of suppliers (adding any arbitrarily large number of new suppliers, should not increase the file size).
III. Given a supplier key (or set of keys), the supplier can only decrypt:
    a) the parts for which he has design or build responsibility,
    b) any child of a part for which he has design or build responsibility down to a part for which another supplier is responsible (exclusively),
    c) any part with which he needs to interface.
IV. The OEM, can decrypt everything.
V. A change in access policy to a part, should not require the transfer of any keys between the OEM and the supplier.
    a) Note that in practice, this will be accompanied with a revision to the part geometry and if the ownership of the part changes from supplierA to supplierB, supplierA will not be able to decrypt the new revision of the part but he can obviously still decrypt the old revision which he owns. Similarly, supplierB will not be able to decrypt the old revision of the part before he took ownership of it.

3 Literature review

A fairly thorough review of the literature didn't produce any schemes that could be directly plugged and solve this problem. The closest work that addresses a somewhat related problem was documented in the paper Group Key Management Scheme for Simultaneous Multiple Groups with Overlapped Membership (Purushothama & Amberker, 2011). The paper describes a scenario where multiple users belong to different groups and can arbitrarily join and leave other groups. The users are stored as nodes in a Key-User Tree (KUT) with one tree per group and keys are generated by a Key Distribution Center (KDC). The users can then send messages to each other and can only decrypt messages from their "parental" group or any group which they joined thereafter. The use-case while similar on the surface is actually quite different. In our case, the equivalent to a group user could be a supplier and the OEM could act as the KDC. The key difference is that the suppliers do not necessarily want to interact with each other and the concept of a group doesn't really exist.

Although the scheme cannot be used as-is, the general idea on which it is based (although curiously not credited in the paper), that of the secret sharing (Shamir, 1979), can actually be adapted to work in this case. In his elegant and beautifully succinct paper, Shamir describes the idea that allows multiple people to share a piece of a secret that can only be reconstructed when at least n - 1 parties come together. The idea is based on the fact that any polynomial of degree n can be obtained from at least n - 1 distinct points. If we have a large group of entities (larger or equal to n - 1), each holding their own point, any set

of *n* - 1 among those entities is sufficient to find the polynomial (crucially, not all points are necessary but any set smaller than *n* - 1 will be insufficient). This polynomial is then intersected against the *y* axis to obtain the key *K* that is necessary for the decryption.

A second paper that dealt with a somewhat similar problem was Polymorphic Encryption and Pseudonymisation for Personalised Healthcare (Verheul, Jacobs, Meijer, Hildebrandt, & de Ruiter, 2016). In this case, data is encrypted and a trusted third party called the transcryptor can tweak the ciphertext so that a specific party with the right secret key can decrypt it. In this case, the key goal is that no one other than the person holding the secret key can decrypt the ciphertext (either in its intermediate stage which is hosted somewhere in the open or in its final stage) but multiple parties can request to get access to the data and get it reencrypted by the transcryptor so that they can decode it. Again, there are some similarities to our use case but the key difference is that we don't have a requirement that the data cannot be decrypted by the host (in our case the OEM) so the whole scheme is redundant since we might as well just encrypt the data for each party right away. This is though specifically what we want to avoid: having to encrypt the data specifically for each supplier.

4 Solutions to the problem

This section explains the incremental construction of the solution.

4.1 Naïve solution: 1 key per part

The most obvious solution is to assign a different key each part and distribute the appropriate keys to each supplier based on the access policy of the part. This approach has a multitude of flaws, such as the facts that the number of keys will become unmanageable, but mainly the fact that each new part or a revision to a part requires the transfer of a new key to a set of suppliers. Crucially, this solution violates requirement V. It also requires the xml file to contain a reference to the key which needs to be used. Not necessarily a big security violation but certainly not very elegant.

4.2 Slightly better solution: 1 key per access policy

This solution is mainly a slightly more efficient variation of the previous one. In this case, if two parts are meant to be decrypted by the same sets of suppliers, they will be encrypted with the same key. While the number of keys to manage is smaller, it still has all the same issues as the previous one.

4.3 Even better solution: 1 key per access policy with key sharing

This is where Adi Shamir's secret sharing scheme comes into play. The idea is that each supplier gets a single 2-dimensional point in a Galois Field. Similarly to the previous method, one secure key per access policy is assigned and the data is encrypted using those keys. Next, a Lagrange polynomial is computed for the points that belong to each of the suppliers in the access policy plus the point (0,key). Finally, a set of points (of size equal to the number of suppliers in the access policy) is computed on that polynomial (different from the supplier owned points) and those points are shared in the encrypted xml file.

Figure 3 - Simple key sharingFigure 3 illustrates the simple case where there are only two suppliers (Polestar and Brembo).

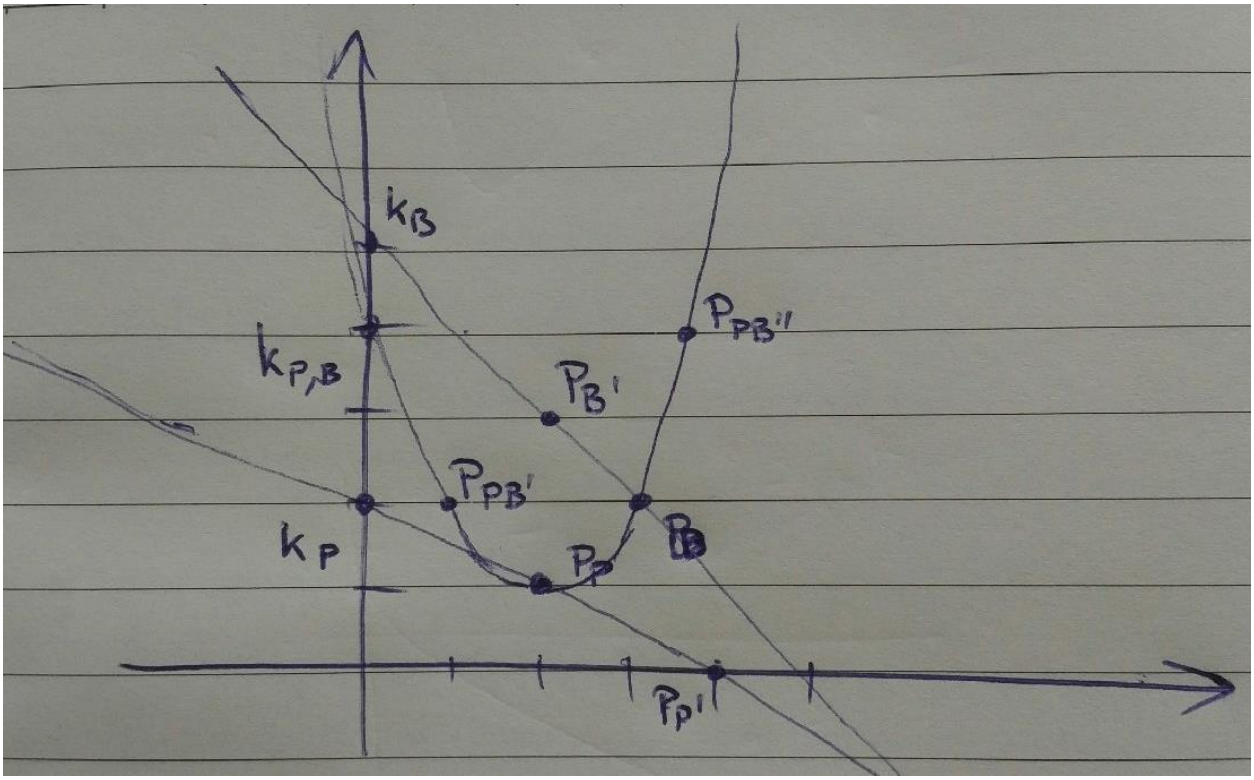

Figure 3 - Simple key sharing

P_B and P_P are the points that are given to Brembo and Polestar respectively at the signing of the contract between the supplier and the OEM. They remain stable for the life of the contract. k_P, k_B and $K_{P,B}$ are keys used for encrypting each part based on its access policy.

- If the part needs to be decrypted by Polestar only, it will be encrypted using k_P.
- If it needs to be decrypted by both Polestar and Brembo, it will be encrypted using $k_{P,B}$.

Finally, the encrypted xml file that is distributed also contains the appropriate key-share points for each $P_{X'}$ again based on the access policy. If the part needs to be decrypted

- If the part needs to be decrypted by Polestar only, the key-share will contain $P_{P'}$.
- If it needs to be decrypted by both Polestar and Brembo, the key-share will contain $P_{PB'}$ and $P_{PB''}$.

Now, let's say that Brembo tries to decrypt three different parts each encrypted with a different key.

- If it has been encrypted using k_B, the key-share will contain $P_{B'}$. All Brembo needs to do it find the Lagrange polynomial $\mathcal{P}_B$ for P_B (which only Brembo has) and $P_{B'}$ (which everyone has) and solve it for (x = 0) to get k_B.
- If it has been encrypted using $k_{P,B}$, the key-share will contain $P_{PB'}$ and $P_{PB''}$. This time, Brembo needs to find the Lagrange polynomial $\mathcal{P}_{PB}$ for P_B (which only Brembo has), $P_{PB'}$ and $P_{PB''}$ (which everyone has) and solve it for (x = 0) to get $k_{P,B}$.
- If it has been encrypted using k_P, the key-share will contain $P_{P'}$. This time, Brembo will attempt to compute the Lagrange polynomial using the points P_B (its secret point), and $P_{P'}$ (which it finds

in the key-share in the xml). Given the this will produce a different polynomial from $\mathcal{P}_P$, solving it for (x = 0) will produce some other key that will fail to decrypt the ciphertext.

In this simple case with only two suppliers, Brembo only knows that Polestar's secret point lies somewhere on the polynomial $\mathcal{P}_{PB}$. If the Galois field is big enough, the number of such points will be too large for Brembo to find it efficiently. This property does not hold though if the number of suppliers is larger than two. Let's say we introduce Bosch into the mix. Now, Polestar and Brembo share two polynomials: one in which Bosch is included and one in which it isn't. All that Polestar or Brembo need to do is to find the intersections of those two polynomials to find possible candidate for the other supplier's secret point. This is obviously not desired.

One approach is to limit the number of possible suppliers in an access policy to 2. This actually is acceptable in this use case since a part is only shared with the supplier who builds it and the supplier who needs to interface with it (although, arguably, it's possible that multiple suppliers could interface with a single part). With that restriction any supplier will always hold at most one polynomial in common with any other supplier and won't be able to intersect anything. This method still presents a security risk though. In a very plausible scenario where two dishonest parties collaborate with each other (two suppliers owned by a single parent corporation or two suppliers owned or controlled by a state actor), all they need to do is share with each other the polynomials they share with another supplier which they can then intersect and find this third supplier's secret point.

A more robust approach is discussed in the following sub-section.

4.4 Final solution: key sharing with salting and hashing

The final solution is a slight variation on the previous one. Instead of computing the keys directly from the secret supplier points, we add a salt and translate the point by applying a secure hash function such as (SHA-512) and then compute the key from those translated points. The salt is computed randomly for each part and included in the xml file. All a supplier needs to do is append the salt, compute the hashes from the concatenations of the salt and the two coordinates, truncate the results and use the newly obtained point and the key-shares following the same method as in the previous sub-section.

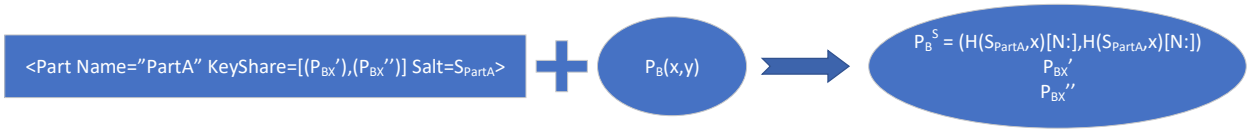

Figure 4 - Salting and Hashing

Given that now a different point is used for each part, intersecting polynomials will not give an attacker anything they can use to decrypt any other part. Using this method any arbitrary amount of suppliers can be included in any access policy without the risk of a dishonest party finding the secret points of other suppliers.

5 Implementation

The implementation used to obtain the results presented in this section follows the scheme described in section 4.3[2]. The following figures show the encrypted xml and the decryption obtained by Polestar.

```
1  <Car AccessPolicy="()" Name="Volvo V70R" Owner="Volvo">
2      <Assembly AccessPolicy="('Polestar',)" Geometry="4979993fb4e1918df4b71192631ad379e488c47b5d435d8173e07dad254d0763"
   KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Front Subframe" Owner="Polestar" Quantity="1">
3          <Assembly AccessPolicy="('Polestar',)" Geometry="0fd38ff09ba5ed6af2125b4cc3d525681ed8a54c5b558a6ee84f54685b0a2a4d"
   KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Front Strut" Quantity="2">
4              <Part AccessPolicy="('Polestar', 'Brembo')"
   Geometry="7ff381d1a15f310e2bcbcf2aa11f94625ac6d8f1cf74a99cae3d2464aee47a21" KeyShare="[(9396, 50163.56608908101), (734,
   37579.05027869038)]" Name="Strut housing" Quantity="1" />
5              <Part AccessPolicy="('Polestar',)" Geometry="5557f6dc3be8da71645703a091402b443108ed6e85be1689d16865959564a0bb"
   KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Upper bearing" Quantity="1" interfacesWith="Chassis" />
6              <Part AccessPolicy="('Polestar',)" Geometry="318747d212e2077c33d0c8ea5ff7b681e488c47b5d435d8173e07dad254d0763"
   KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Shock absorber" Quantity="1" />
7              <Part AccessPolicy="('Polestar',)" Geometry="8eb9532d5d326637c317e51f0ab99f2d" KeyShare="[(47595, 38575), (5342,
   51982.48234263742)]" Name="Spring" Quantity="1" />
8              <Assembly AccessPolicy="('Brembo',)" Geometry="ea0a67e56a6038b0504ebb53248692a6f3d563b1426847d83d7ae9d34333208a"
   KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Brake caliper" Owner="Brembo" Quantity="1">
9                  <Part AccessPolicy="('Brembo',)" Geometry="ba9b5230fce0738e3435684d30cb55fbdd173db5a6f47afa310966859ed9ecaa"
   KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Pad wear sensor" Quantity="1" />
10                 <Part AccessPolicy="('Brembo',)" Geometry="5299175f9eb579515a337cec34505225f3d563b1426847d83d7ae9d34333208a"
   KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Bleeder screw" Quantity="1" />
11                 <Part AccessPolicy="('Brembo',)" Geometry="9f6d0495f8ad5fca458a878980172bc9f2b653fff17dad7e8662f3df06b2044c"
   KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Caliper body" Quantity="1" interfacesWith="Strut housing" />
12                 <Part AccessPolicy="('Brembo',)" Geometry="67d659eec8e60dd7eb1c27fdba971c5a79ab548277cd68455e8048f728f23629"
   KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Anti-rattle clip" Quantity="1" />
13             </Assembly>
14         </Assembly>
15         <Part AccessPolicy="('Polestar',)" Geometry="664da5c5ce8587379010733476b43cb4e488c47b5d435d8173e07dad254d0763"
   KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Stabilizer bar" Quantity="1" />
16         <Part AccessPolicy="('Polestar',)" Geometry="1c7044d122223a41f0d5073565ec354f78abe10e87c4a7a126c7b5046a48e93c"
   KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Bushing" Quantity="4" />
17     </Assembly>
18     <Part AccessPolicy="('Polestar',)" Geometry="5b7aaf4776b2f28c5e31f59a5385885578abe10e87c4a7a126c7b5046a48e93c"
   KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Chassis" Quantity="1" />
19     <Part AccessPolicy="()" Geometry="1222fc9725b69ad658436b4962e318858b99deb00dd00cb2f9c6b2643657ac99" KeyShare="[(27565,
   48338), (35001, 34822)]" Name="Rear Subframe" Quantity="1" />
20 </Car>
```

Figure 5 - Encrypted xml file

Figure 6 shows that the Geometry attributes which contain Polestar are back to their respective plaintexts. All the other Geometry attributes are empty. The way padding is implemented, it removes *x* last characters for *x* equal to the last byte without validating that the pad is actually correct. In this case, the last bytes were larger than the full length of the obtained plaintext. While this is obviously insecure, as it directly leaks the value of the last byte for sufficiently long strings, implementing a secure padding scheme was not the goal of this project.

[2] I wrote the implementation before waking up with the final solution in my brain at 3:21 AM the night before the deadline (true story) and unfortunately ran out of time to fully implement the revised scheme.

```
<Car AccessPolicy="()" Name="Volvo V70R" Owner="Volvo">
    <Assembly AccessPolicy="('Polestar',)" Geometry="Front Subframe geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Front Subframe" Owner="Polestar" Quantity="1">
        <Assembly AccessPolicy="('Polestar',)" Geometry="Front Strut geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Front Strut" Quantity="2">
            <Part AccessPolicy="('Polestar', 'Brembo')" Geometry="Strut housing geometry" KeyShare="[(9396, 50163.56608908101), (734, 37579.05027869038)]" Name="Strut housing" Quantity="1" />
            <Part AccessPolicy="('Polestar',)" Geometry="Upper bearing geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Upper bearing" Quantity="1" interfacesWith="Chassis" />
            <Part AccessPolicy="('Polestar',)" Geometry="Shock absorber geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Shock absorber" Quantity="1" />
            <Part AccessPolicy="('Polestar',)" Geometry="Spring geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Spring" Quantity="1" />
            <Assembly AccessPolicy="('Brembo',)" Geometry="" KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Brake caliper" Owner="Brembo" Quantity="1">
                <Part AccessPolicy="('Brembo',)" Geometry="" KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Pad wear sensor" Quantity="1" />
                <Part AccessPolicy="('Brembo',)" Geometry="" KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Bleeder screw" Quantity="1" />
                <Part AccessPolicy="('Brembo',)" Geometry="" KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Caliper body" Quantity="1" interfacesWith="Strut housing" />
                <Part AccessPolicy="('Brembo',)" Geometry="" KeyShare="[(30142, 17574), (13958, 36221.155282078325)]" Name="Anti-rattle clip" Quantity="1" />
            </Assembly>
        </Assembly>
        <Part AccessPolicy="('Polestar',)" Geometry="Stabilizer bar geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Stabilizer bar" Quantity="1" />
        <Part AccessPolicy="('Polestar',)" Geometry="Bushing geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Bushing" Quantity="4" />
    </Assembly>
    <Part AccessPolicy="('Polestar',)" Geometry="Chassis geometry" KeyShare="[(47595, 38575), (5342, 51982.48234263742)]" Name="Chassis" Quantity="1" />
    <Part AccessPolicy="()" Geometry="" KeyShare="[(27565, 48338), (35001, 34822)]" Name="Rear Subframe" Quantity="1" />
</Car>
```

Figure 6 - xml file as obtained by Polestar

In the spirit of full disclosure, the implementation has multiple issues which I did not focus on for the purpose of this project. My goal was to verify the approach and show that it can work. A robust implementation would require a lot of additional effort.

- It is plagued with precision errors. The decryption actually only works when the precision errors do not affect the evaluation of the polynomials (the intersections with the y-axis are rounded to the closest integer). Because of that also, the prime numbers selected are very small.
- It does not actually implement a proper Galois Field.
- Nothing is done to ensure that the points and coefficients are integers. While this is fairly straight forward to do, it is also time consuming.
- I'm sure I'm missing some more…

6 References

Purushothama, B. R., & Amberker, B. B. (2011). Group key management scheme for simultaneous multiple groups with overlapped membership. *Third International Conference on Communication Systems and Networks.* Bangalore: IEEE.

Shamir, A. (1979). How to share a secret. *Communications of the ACM*, 612-613.

Verheul, E., Jacobs, B., Meijer, C., Hildebrandt, M., & de Ruiter, J. (2016). *Polymorphic Encryption and Pseudonymisation for Personalised Healthcare.* Nijmegen, the Netherlands: Institute for Computing and Information Sciences, Radboud University.