International Journal of Cyber-Security and Digital Forensics (IJCSDF) 9(2): 102-112
The Society of Digital Information and Wireless Communications (SDIWC), 2020 ISSN: 2305-0011

Blueprint Key: A Tool for a Novel Layer of XML Encryption

Jason Meza and Paolina Centonze
Iona College
9 Bayard St, New Rochelle, NY, 10805
pcentonze@iona.edu

1 ABSTRACT

In this paper, we expand upon the robust security mechanisms of XML data. There are many methodologies for XML security, such as XML encryption and XML signature, which are W3C recommendations. Although these methods meet the XML security standard, they do not signify that any XML data is secure or unbreachable. XML vulnerabilities are among OWASP's top-ten security risks by XML External Entity or XXE attacks. Therefore, many researchers have devised their mechanisms to increase protection for XML data, by either improving known methods or creating their own to improve XML security from attackers. Combining methods is also an option, creating layers of protection. Even if an attacker can penetrate one layer, another layer creates a stricter barrier for the attacker to achieve any malicious attack. Blueprint Key (BK) is our novel XML method to provide that extra layer of protection. BK separates XML data from structure, making the structure as important as the content. What makes BK so unique is the fact that the key itself is another XML file. Results show that encryption and decryption speed are very minimal and that adding it to any process increases security for XML data.

2 KEYWORDS

XML, XML Encryption, XML Signature, Steganography, Asymmetric Encryption, Symmetric Encryption, Caesar Cipher, Vigenère Cipher.

3 INTRODUCTION

XML is a data structure applied across many platforms and frameworks. An XML file may contain sensitive data such as personal, account, statement, or banking information. Thus, XML security enforcement is essential to ensure the protection of any sensitive data. The World Wide Web Consortium (W3C) XML security standards are XML encryption and XML signature. XML encryption protects the content of a file while XML signature serves to maintain authentication or integrity. Both methods deal with applying special XML elements, such as EncryptedData or SignatureInfo, performing their desired function. Many researchers have their algorithms, concealing, or reconstructing data unrecognizable or unreadable through encryption. Therefore, implementing known methods such as Caesar or Vigenère Cipher is not uncommon. However, these techniques, as well as XML encryption and signature, have been defeated before. They are exposing weakness in XML, leading to XML security containing vulnerabilities. Considering XML is a widely used data structure, recognizing the vulnerabilities as an issue merited a rank on OWASP's top-ten security risks. Hence, it is crucial to implement and update XML security methods to ensure sensitive data is protected.

Blueprint Key (BK) is our proposed method to secure the XML structure by rendering it unrecognizable. Most encryption methods contain a central algorithm, math techniques, and other data structures to reconstruct or hide data. With BK, the central algorithm or key is another XML file where the original structure or specific data have their particular locations marked on the BK XML file. This makes the BK File key crucial in encrypting and decrypting any file regarding BK methods. BK's main positive

effect is separating data from structure or making the data and structure unrecognizable from each other. Merging techniques may increase security but also take a toll on speed. Understanding the importance of speed vs. security is essential in order to obtain the proper balance. The primary purpose of BK is to enhance an XML security process. In doing so, BK becomes merely another security layer which serves to increase the protection of XML data. Steganography is the practice of concealing information such as a message, image, or audio within a separate file to avoid detection. BK is a form of steganography in that it hides XML structure and content using a separate file. What makes BK so unique is its crucial reliance on another XML file. In this paper, we discuss past methods along with their advantages and their disadvantages. We explain the benefits and risks of Blueprint Key methods as well as how to utilize it, as there are different approaches for doing so. We then describe our results and make comparisons to past works, concluding with summarizing how Blueprint Key can be a great addition to any XML security process.

4 PAST METHODS

Taflan I. Gündem, et al. in [1] introduce their technique, XML structure encryption. The general idea here is to possess two keys; one to encrypt elements and values and another to encrypt the XML structure. The structure can be encrypted by using a relational schema and tables to record the information. The field key algorithm uses CBC mode, meaning two similar elements, like a parent ID, can be encrypted to have different values. Another method would be to create a structure ID or str-ID. Decrypting would require this information to reconstruct and obtain the original structure. Doing so would indicate that the recorded information is the structure key, while any other added encryption method for the content would indicate another key. This causes a separation of content from structure, halving the entire original file. If the content key was a known method that had been breached before, such as XML encryption, then an attacker would still need the structure information to have the entire file. XML structure encryption makes the structure of an XML file just as crucial as its content. Therefore, protection for both is needed, which in the end requires two keys: the structure key and the content-encryption key.

Keiko et al in [2] explains in-depth XML encryption and symmetric encryption. Symmetric encryption uses the same key for both encryption and decryption. A sender uses a function (E) and a key (k) to encrypt the message (M) to produce the ciphertext (C).

$$C = Ek(M) \quad (1)$$

The receiver then takes the ciphertext(C), a function (D), and the same key (k) to obtain the message (M).

$$M = Dk(C) \quad (2)$$

XML encryption supports symmetric encryption, in which XML elements are involved in containing the encryption method and properties such as the key. The data to be encrypted is usually serialized beforehand, such that the data is converted into octlets. Then, after choosing an algorithm and a key, the cipher data is then in XML format. This procedure is all carried out in XML encryption, but the principal of symmetric encryption is the primary intent. Therefore, the same benefits and weaknesses of symmetric encryption apply to XML encryption. For instance, one advantage is that any user who possesses the key can encrypt and decrypt as they please. Of course, this is also a disadvantage in that an attacker who obtains the key may perform malicious activities to the XML. Evidently, XML encryption isn't completely secure, thus, implementing our BK method after XML encryption is a great addition that serves to increase XML security.

A. A. Abd El-Aziz, et al. in [3] create their own roadmap of XML signature and encryption for future work and research. They illustrate how XML signature can be a solution for falsification while ensuring authentication and integrity. XML signature has various different signature approaches such as detaching signature, enveloped signature, and an enveloping signature. A detached signature refers to having data separated by signature as opposed to having it all in one document. An

enveloped signature refers to a signature being a part of the document. An enveloping signature refers to signed data being contained within itself. XML signature is suitable for distribution in the network ecosystem. A combination of XML signature and encryption has been carried out before but has also been broken in the past.

N. Nithin, et al. in [4] address the drawback of computation time in RSA asymmetric encryption algorithms, proposing a new encryption method for XML files called XML Batch Multi-Prime RSA (XBMRSA). Asymmetric cryptography has two keys, a public and private key. The public key is well known, meaning many users can produce their own ciphertexts, whereas the private key is known only to few and is used to decrypt the secret information. Asymmetric RSA deals with choosing two prime numbers, preferably at random to ensure security. It then employs a module function that is used for decryption as well. In XBMRSA, for any integer $r \geq 2$, key generation is defined by letting N be a product of r, which is randomly chosen from distant primes p1, p2, ... pr. Thus, the pair (N, e) is used for the public key, while (N, d) is used for the private key. This results in the same equation for encryption and decryption for RSA. When $r = 2$, only two prime numbers are used, much as in the original RSA algorithm. However, when $r > 2$, there are multiple primes in the modulus N. This r creates the difference between RSA and XBMRSA. The purpose of XBMRSA is to decrease the drawback of computation time in RSA. After testing with multiple XML files, the computation time was decreased and more efficient than with RSA.

Ankita, et al. in [5] describe how they improved the security in Service Oriented Architecture (SOA) in web service using XML encryption and Signature. They explain the concept of XML encryption units such as the whole document, one element, one element value, or binary value outside of the XML document. Using these different units and the method itself improves upon the security of the message. Much like XML encryption, XML signature also uses elements, thus allowing an encrypted file to be signed for authentication and integrity. They also mention an XML Certificate, which also improves upon the security of a message. It consists of the certificate name, version, serial number, and validity. They give an overview of how W3C standards can be used to secure a message system. There are many instances and is much research on how to break the standards in the W3C recommendations, such as in [6] and [7]. Both show how these elements for XML encryption and XML signature can be exploited. This is due to how flexible XML can be in general. XML typically contains elements such as nodes, specifically text element nodes, and attributes. There are many ways to utilize these elements using our proposed BK method.

Alberto, et al. in [8] propose a security framework for XML schemas. They mainly focused on healthcare information to facilitate the exchange of this information via the Continuity of Care Record (CCR). CCR is used for storing and managing medical tests, scans, diagnoses, and/or patient data. Their objective was to have information displayed differently depending on a user's authorization, thus generating a customizable access control security tool. This policy can also be enforced during runtime to ensure that the correct data is displayed and is secure. This method is similar to that of our BK in that the schemas need to recognize elements using identifiers. Identifiers are necessary to use the schemas and present the data correctly to the user's permission. This research is a fair comparison for BK in a real-world application.

Ari, et al. in [9] present a model prototype of how to include security in the exchange of data. A message sent by the web service can be altered so that the structure is changed at the risk of not even being sent at all. For this study, the researchers built a system comprising of a web service that uses XML encryption and RSA cryptography. They designed it so that every request is authenticated and confidential and the XML data communicating between the client and the webserver is safe (encrypted). Decryption occurs when data is requested by a user using the private key and public key with RSA. Results showed that their design and implementation addressed the security issued with authorization, authentication, and confidentiality. However, the key exchange

was limited; therefore, the key exchange needs more protection.

V. Shanmughaneethi, et al. in [10] explain how XPath injection, which is similar to SQL injection, can be a useful technique employed by an attacker to obtain some information and perform a successful attack. They're proposing an approach to detect XPath injections during runtime by intercepting and parsing expressions for inputs. Similar to SQL injection counters, strong input validation, parameterized queries, and custom errors are common ways to prevent XPath injection. But these methods aren't sufficient to be fully secured against XPath injection. Their proposed method therefore includes an XPath Injection Vulnerability Detector (PXpathV) so that any inputs run through the module are also evaluated with the XPath Expression Scanner. Using the scanner during runtime allows any generated queries to be intercepted before they reach the database server. The next module is the XPath Expression Analyze module which detects vulnerabilities. After catching a query, the analyzer stores them in an XML document which is used to test for vulnerabilities. The last module is XQuery validation, one of the most effective techniques, which uses XML schema to define and validate an XML document's structure. They found that the time difference with or without the inclusion of the PXPathV module was very minimal. Due to the possibilities without the PXPathV module, they found their methods to be successful such that vulnerabilities are breached in no time.

5 PROPOSED METHOD: BLUEPRINT KEY

Blueprint Key (BK) is our proposed method, its primary purpose a flexible addition to an encryption process. The inspiration for BK was derived from a combination of XML structure encryption (XMLSE) and steganography. It encrypts the crucial structure information while hiding the contents using another XML file. BK creates another key much like in XMLSE, such that the original file is split into two different ones. This data is recorded within the BK file itself, which is therefore required to retrieve the actual information. The encrypted file has the actual content or hidden structure. The BK file serves as the blueprint of the original file and contains the instructions to obtain the original function.

There are many instances and is much research on how to break the standards in the W3C recommendations, such as in [5] and [6]. Both show how these elements for XML encryption and XML signature can be exploited due to how flexible XML can be in general. XML typically contains elements such as nodes, specifically text element nodes, and attributes. There are many ways to utilize these elements using our proposed BK method. Here we outline two methods for doing so when encrypting or recording the XML file structure and content while we are halving the original file to create the BK key file and the encryption/decryption file.

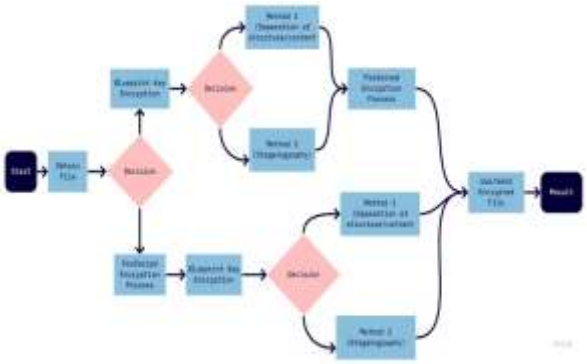

Figure 1: Adding BK into the encrypting process leads to the same result.

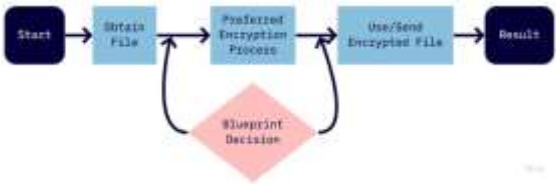

Figure 2: Simplified decision making for BK.

Figure 1 shows that you can decide when to use the BK method at any point and it would produce the same result. Adding other factors is also possible. Figure 2 is a simplified process compared to Figure 1, where adding BK is as simple as including it in either a step before or after encryption.

Figure 3: Method 1, separating the XML data from structure.

Method 1, which is similar to XMLSE, extracts and encrypts both the content and structure, resulting in two keys or recorded information. The encrypted file then serves as a file containing identifiers or symbols that pinpoints the location of the content. The structure key and BK file are integral for placing the content. Without the content information, there wouldn't be any valuable information. As you can see in Figure 3, Method 1 separates data from structure. This makes structure just as important as the content. Thus, two keys are created instead of one when encrypting and decrypting.

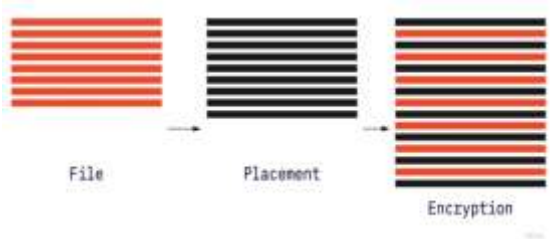

Figure 4. Method 2 (steganography) hiding the file in another file.

Method 2 hides the XML file in another XML file. This can be done by adding random data around the actual content or moving the elements to different locations in the placement file. Therefore, the BK file identifies the location of the actual content. In Figure 4, the file contents marked in red are hidden in the placement file. Just as with steganography, the content is hidden within other content. This is a form of BK encryption, where the Blueprint Key file is needed in order to determine which data is the correct data. This makes it possible to use the placement data as an indicator if any data has been used. For example, if the placement file contains its own indicators, any leaked data may indicate to us if it's placement or the actual data being used.

Both methods are flexible when using identifiers. Texts, attributes, or the nodes themselves can indicate where the identifiers are placed—creating multiple ways to have the data and structure recorded. Both methods deal with an encrypted file, yet method 1 has the data hidden somewhere in the contents of another file while method 2 has the contents removed and recorded. Regardless, both methods require the BK file to be either mapped to the contents of the original file or located in the same place as the original contents. In addition, BK can have different keys (files) mapped to the desired file as well as choose the placement file that will hide the original file.

Figure 5. Example of an attacker.

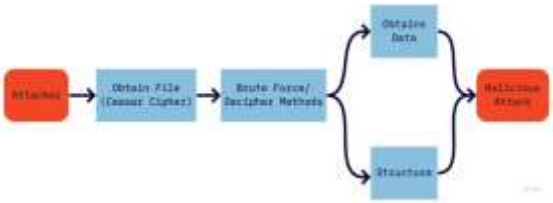

Figure 6. Example of how splitting data from structure limits the attacker from succeeding.

Both methods allow the content to be randomized, ensuring that the structure cannot be traced back without the BK key file. Figure 5 demonstrates a simple example of how an attacker can succeed in an attack. While Figure 6 demonstrates this with the same process, the attacker can still launch an attack but only with half the data. The attacker will need to continue in order to get the original file.

```
<students>
    <student id ="one">
        <name>Jason</name>
        <last>Meza</last>
    </student>
    <student id ="two">
        <name>Alex</name>
        <last>Quispe</last>
    </student>
    <student id ="three">
        <name>Sal</name>
        <last>Alzubidi</last>
    </student>
    <student id ="four">
        <name>Joey</name>
        <last>Viera</last>
    </student>
```

Figure 7. An example of a test XML file's content.

Figure 7 shows how the test file contents are being presented. It is a simple XML file that contains basic student information, such as first and last name. All the test files contain similar content as the most important aspects are the size of the files as well as how the data is being manipulated.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<students>

    <KOSC2/>

    <AWBSP/>

    <TU3ZT/>

    <QDMC1/>

    <DJ63Q/>
```

Figure 8 An example of how a Blueprint Key file would look.

Figure 8 is an example of how an XML Blueprint key file would look. The root element "students" is noticeable, as only students are the target. If the target was a more specific element such as first or last name, then the key would appear to have the "student" elements but have the identifier in the place of the desired target.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<students>

    <student id="one">

        <XXES/>

        <last>Meza</last>

    </student>

    <student id="two">

        <LXSQO/>

        <last>Quispe</last>

    </student>

    <student id="three">

        <KEST/>

        <last>Alzubidi</last>

    </student>
```

Figure 9. An example of the encrypted Blueprint file.

Figure 9 shows an example of how the first name, the desired target, is removed from the file, but replaced with an identifier for that specific node. These symbols can also be randomized, so that if an attacker obtains the extracted information key, they would need the Blueprint key file in order to make sure the correct data is placed into the correct location.

```
/*****************    NODE FUNCTIONS    **********************/
public void thruNode(Node copy, Element previous) {
    previous = setAttr(copy, previous);

    NodeList nlist = copy.getChildNodes();
    for(int i = 0; i < nlist.getLength(); i++) {
        Node current = nlist.item(i);
        if(current.getNodeType() == Node.ELEMENT_NODE) {
            Element next = setEle(current, previous);
            thruNode(current, next);
        }
        if(current.getNodeType() == Node.TEXT_NODE) {
            previous.appendChild(encDoc.createTextNode(current.getTextContent()));
        }
    }
}

public Element setAttrNode copy, Element previous) {
    if(copy.hasAttributes()) {
        NamedNodeMap map = copy.getAttributes();
        for(int i = 0; i < map.getLength(); i++) {
            Node current = map.item(i);
            String s1 = current.getNodeName();
            String s2 = current.getNodeValue();

            previous.setAttribute(s1, s2);
        }
    }

    return previous;
}

public Element setEle(Node copy, Element previous) {
    Element ele = null;

    if(elements[0]) {
        if(copy.getNodeName().toString().equals(target)) {
            ele = encDoc.createElement(clone.get(0).keyinfo);
            previous.appendChild(ele);
            clone.remove(0);
        }
        else {
            ele = encDoc.createElement(copy.getNodeName());
            previous.appendChild(ele);
        }
    }
    else {
        ele = encDoc.createElement(copy.getNodeName());
        previous.appendChild(ele);
    }

    return ele;
}
```

Figure 10. Sample code of how to parse through an XML document, inserting the key identifiers into the encrypted file once the target element has been found.

Figure 10 displays the code for how the XML data is read using recursion methods to parse each node. In the function setEle(), if

the element target are nodes, the code checks if the node name equals the target element. If it does, then this code inserts the keyinfo representation in that targets' place, consequently placing that specific information location in the structure. There are multiple ways to read through XML data and Figure 1 is but one of many.

```
public void setInfo(NodeList nlist) {
    sym.setCand(5, true, true);
    ArrayList<String> symbols = new ArrayList<String>();

    while(symbols.size() != nlist.getLength()) {
        String str = sym.generateSym();
        if(!symbols.contains(str) && !Character.isDigit(str.charAt(0)))
            symbols.add(str);
    }

    for(int i = 0; i < nlist.getLength(); i++) {
        Node current = nlist.item(i);
        KeyNode kn = new KeyNode(current, symbols.get(i));
        info.add(kn);
    }
}
```

Figure 11. Sample code of saving the XML content in an ArrayList.

Figure 11 mostly relates to Method 1 which separates the structure and the content. This function, setInfo(), uses a class called KeyNode and an ArrayList to record the information. The KeyNode is a class that has a node and a string. Thus, we can create different strings, and while we capture target nodes, create KeyNode to map string symbols to the node representation. Method 2 is simply extracting information which can be done however the developer pleases. At the end, the BK file key contains these symbols/identifiers that pinpoint the original location of said target node.

This separation of structure and content is what makes BK a flexible proposed method. There are different ways to carry out this method, but the most important part is determining how to map the structure with the content using the Blueprint Key file.

6 TESTING

The test is done using Java language, using w2c packages such Element, Node, NodeList and Document. While parsing through XML files, the content is recorded using an ArrayList caring a custom class called KeyNode. While parsing through the XML data, once a target node, attribute, or text element is found, it is replaced with a random generated symbol. This is also placed within the Blueprint Key file. The outcome is a Blueprint encrypted file, which still has some data information, depending on how far the data was extracted. Since BK deals with parsing the entire file to create both the key and encrypted file, we tested XML files up to 1.5MB in size with Caeser and Vigenère Ciphers. Both these methods are well known and each are used to encrypt every XML element with the testing XML data.

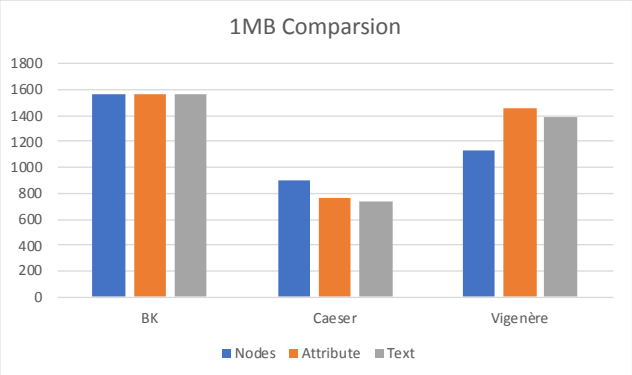

Table 1. A 1MB Comparison encryption time between Blueprint, Caeser, and Vigenère ciphers.

According to Table 1, BK encryption speeds are similar to that of Vigenère, but Caeser provided much faster speeds when dealing with encrypting the file. The decryption speed for these files are closely related to encryption speed; therefore, this chart resembles decryption speed as well as encryption speed. Using different element approaches for BK did not affect the encryption speed at all, whereas for the other two methods, there is variance when using the different elements.

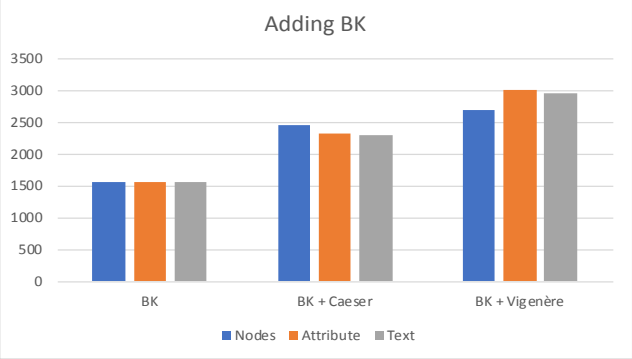

Table 2. A 1MB Comparison encryption time between Blueprint alone, BK + Caeser, and BK + Vigenère.

According to Table 2, adding the BK method to the known Caeser and Vigenère methods causes the number of milliseconds to at least double by comparison to the un-layered methods. Evidently, layering with BK increases the amount of time needed to encrypt, which is to be expected as BK serves as an additional process. This feat requires that an attacker need even more time to breach through the processes with BK, not only because it acts as an additional layer, but also because BK is a new method.

Table 1: Evaluation of Structural XML Encryption Algorithm

EVALUATION OF THE STRUCTURAL XML ENCRYPTION ALGORITHM	
Size of the Document	Time to encrypt/decrypt (seconds)
1 KB	0,4
2 KB	1,2
4 KB	2,1
8 KB	4,9
16 KB	8,3
64 KB	32
256 KB	136,5

12

1 MB	543

Table 3. Results from [1] determined using XML structure encryption.

Table 3 is from [1] wherein the authors shared their results when implementing their method and the corresponding response times.

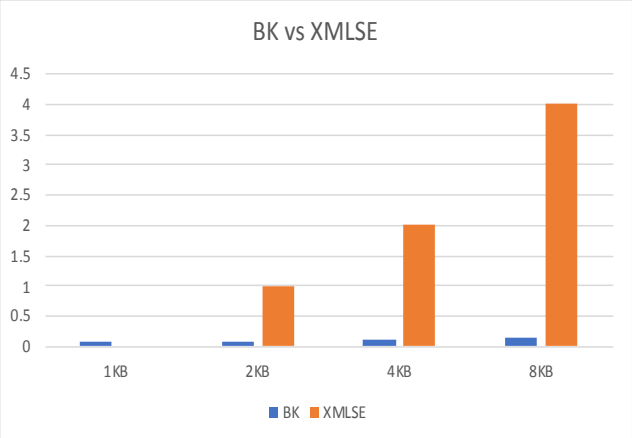

Table 4. A comparison between Blueprint and XML structure encryption.

When comparing the Blueprint results (in milliseconds) to the XML structure results in Table 3 (in seconds), a vast difference is noticeable as the file size gets larger, which is evident in Table 4. This difference is due to how the structure is handled. In [1] they create Stc-ID fields and then have to record the data with the content reference. They have to take a larger portion of the file in order to completely extract the information, encrypt it, and then produce an encrypted file. With Blueprint Key, the structure is represented with the key file. Therefore, the Blueprint process recorded less information while still maintaining the structure to content relationship despite the separation. This shows the benefits of using the Blueprint key method of recording the structure instead of recording all information with the content id in a table as well.

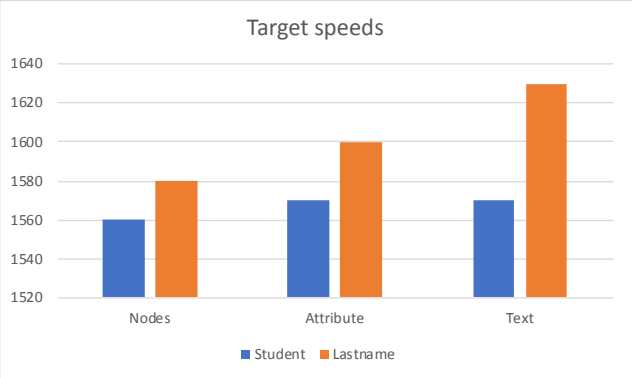

Table 5. A target speeds comparison boasting a larger content target vs a specific lower content target.

The test not only dealt with file size but also with target encryption. The test files

all contained student information for the example test. The target was either the entire student information node or only their last name. According to Figure 3, using the entire student information node as the target was a lot faster than using only their last name as the target. This is due to the search value of the targets. For instance, the student node occurd first when parsing though the XML file. The last name, on the other hand, is a deeper node when searching through the file, therefore it stands to reason that it takes longer to search for a deeper level node within an XML structure.

The manner in which the Blueprint Key can be created relies on how flexible access to the elements are. There are many languages which have frameworks that support XML and give various ways to parse XML data. Comparing Blueprint with the known Caesar and Vigenère Ciphers can show how the speed of encryption time remains more or less consistent using our method. Comparing it to the method of XML structure encryption results from [1], the time to do this takes way less since the structure blueprint is recorded on the XML Blueprint key file.

Using the well-known Caeser and Vigenère ciphers was for speed comparison. The end result was Blueprint encryption taking longer milliseconds to both ciphers. This is due to the file changing the representation of the elements. Blueprint Key has the same outcome but also takes time to record the data. Comparing results with [1] where we found the results to be a big difference, [1] recorded their time in seconds, while our Blueprint results were in milliseconds. This could be due to the difference in testing approaches but the method for [1] also used relational DTD. Their field key algorithm always used CBC mode, creating different outcomes for the same values. But with an evaluation of comparison speeds, Blueprint key was still far slower than results from [1]. This shows that the difference between recording the data structure with another file instead of recording the data can make a difference as the file size increases.

7 CONCLUSION

The known standards for XML security are W3C recommendations XML encryption and XML signature. The longer a method has existed and been implemented, the more chances and time an attacker has to break it. Therefore, new or improved methods must be created and performed—new steps or combinations for an encryption process. In the past methods stated, we learned how the W3C standards are still being used but others are implanting improved or newer methods. Blueprint Key (BK) acts as this extra step and, in doing so, adds an additional layer of security when protecting an XML file. It proves to add more robustness whether the data is first encrypted and then BK methods are used, or vice versa. BK serves as a great addition because it is such a flexible one. The inspiration for our newly proposed BK stemmed from steganography research, as well as [7]. Our proposed Blueprint Key method works as follows: it regards XML structure as crucial information and, as a result, hides it in another XML file. In addition, a BK file is also generated which is required to decrypt or reconstruct the original file. Blueprint key is as flexible as XML itself, given that this method can be done in many different ways, using the XML elements as an advantage. As well as separating content from structure, this provides more complex methods for an attacker to break through. Results show that as the file size increases, so too does the process speed. But seeing how minimal the process speed is, the use of Blueprint Key as a layer won’t significantly add to the overall process speed. Thus, it is ultimately another key or another layer of protection that can be used to protect XML data.

8 FUTURE WORK

Future work could focus on finding an efficient way to use identifiers to record the placement and location of the original data. It could also focus on finding a different way of recording the data with another identifier method or finding another file or medium for disguising the file. Blueprint Key doesn't have to be the primary or lone method used, but it should be the go-to method employed when trying to increase the security or protection of an XML file (which contains sensitive data) in

an attempt to extend our work. Blueprint Key method has plenty of potential, if not to be improved but to inspired new ways for XML Security. We hope that Blueprint Key inspires new and innovative ways to protect XML files.

9 ACKNOWLEDGEMENTS

Thank you, Dr. Centonze, for all your teachings and advice throughout every course and this entire process.

Thank you, Dr. Bailie, for helping me decide where to enroll after completing my Associate Degree. You assured me to believe in Iona College, and I am glad I chose to attend that Spring.

Thank you, Ileana Palesi, for helping me organize my thoughts on paper better than I could by myself. I'm truly grateful for the time and effort that you put into my work.

Thank you, Iona Computer Science Dept facility, for creating a comfortable yet challenging environment as I continue to learn how to kickstart my future career.

Thank you to my Brothers, who know I can be isolated, but know when to be there for me when I need them. I may not always show my appreciation, but know I appreciate everything you guys do for me.

And finally, thank you to my family, for putting up with my nonsense but did so because they believe in what I can do. I know it's not easy to deal with my moments, but I know they wouldn't have me any other way.

10 REFERENCES

1. Gündem, Taflan I, and Mustafa F Çelikel. "STRUCTURE ENCRYPTION IN XML." https://www.citationmachine.net/mla/cite-a-journal/manual.https://www.cmpe.boun.edu.tr/~gundem/xml-encrypt.pdf

2. Hashizume, Keiko, and Eduardo B. Fernandez. "Symmetric Encryption and XML Encryption Patterns." *Proceedings of the 16th Conference on Pattern Languages of Programs - PLoP 09*, 2010, doi:10.1145/1943226.1943243. https://www.researchgate.net/publication/228855793_Symmetric_encryption_and_XML_encryption_patterns

3. El-Aziz, A. A. Abd, and A. Kannan. "A Comprehensive Presentation to XML Signature and Encryption." *2013 International Conference on Recent Trends in Information Technology (ICRTIT)*, 2013, doi:10.1109/icrtit.2013.6844276. https://ieeexplore.ieee.org/document/6844276

4. Nithin, N., and Anupkumar M Bongale. "XBMRSA: A New XML Encryption Algorithm." *2012 World Congress on Information and Communication Technologies*, 2012, doi:10.1109/wict.2012.6409141.https://ieeexplore.ieee.org/document/6409141

5. Journal, Ijert. "IJERT-A Review On Various Message Security Services Using XML." *International Journal of Engineering Research and Technology (IJERT)*, www.academia.edu/44008489/IJERT_A_Review_On_Various_Message_Security_Services_Using_XML.

6. Jager, Tibor, and Juraj Somorovsky. "How to Break XML Encryption." *Proceedings of the 18th ACM Conference on Computer and Communications Security - CCS 11*, 2011, doi:10.1145/2046707.2046756. https://www.nds.ruhr-uni-bochum.de/media/nds/veroeffentlichungen/2011/10/22/HowToBreakXMLenc.pdf

7. Kupser, Dennis, et al. "How to Break XML Encryption – Automatically." https://www.usenix.org/system/files/conference/woot15/woot15-paper-kupser.pdf. https://www.usenix.org/system/files/conference/woot15/woot15-paper-kupser.pdf

8. De la Rosa Algarin, Alberto, et al. "A Security Framework for XML Schemas and Documents for Healthcare." *2012 IEEE International Conference on Bioinformatics and Biomedicine Workshops*, www.academia.edu/3375529/A_security_f

ramework_for_XML_schemas_and_documents_for_healthcare

9. Muzakir, Ari, and Usman Ependi. "THE SECURITY MODEL FOR DATA EXCHANGE USING XML ENCRYPTION AND SECURITY TOKEN IN WEB SERVICE." *ICIBA2013, the Second International Conference on Information Technology and Business Application*, 22 Feb. 2013.

10. Shanmughaneethi, et al. "PXpathV: Preventing XPath Injection Vulnerabilities in Web Applications." *International Journal on Web Service Computing*, vol. 2, no. 3, pp. 57–64, www.academia.edu/37521672/PXpathV_Preventing_XPath_Injection_Vulnerabilities_in_Web_Applications?email_work_card=view-paper