



KALASALINGAM
ACADEMY OF RESEARCH AND EDUCATION
(DEEMED TO BE UNIVERSITY)

Under sec. 3 of UGC Act 1956. Accredited by NAAC with "A++" Grade



Anand Nagar, Krishnankoil, Srivilliputtur (Via), Virudhunagar (Dt) - 626126, Tamil Nadu | info@kalasalingam.ac.in | www.kalasalingam.ac.in

SCHOOL OF COMPUTING

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MINI PROJECT

IT DATA SECURITY – 213CSE4309

STUDENT ASSESSMENT RECORD

Title of the Project : Post Quantum Cryptography Algorithms

Slot 1

Section Number : S - 23

Team No : 12

Members of the Team : 4

S. No	Name of the Student	Register Number	Signature of the Student
1.	K. BHAVESH SHANKAR	99220040083	
2.	S. IMRAN	99220040204	
3.	CH. GOPI KRISHNA	99220040367	
4.	T. JANI REDDY	99220040356	

Signature of the Examiner with Date

S. No	Name of the Content	Page No
1.	Abstract	3
2.	Introduction	4
3.	Objective	5
4.	Flow chart	6
5	Methodology	7
6	Modules	8
6.	Background Study	9
7.	Implementation	10-13
7	Output	14
8.	Future Scope	15
9.	Conclusion	16
10.	References	17

ABSTRACT:

The goal of this project is to put into practice a condensed form of the post-quantum cryptography-specific SPHINCS+ hash-based signature scheme. SPHINCS+ is a stateless, safe digital signature system that is impervious to quantum computer attacks.

It achieves robust cryptographic guarantees by combining FORS (Forest of Random Subsets) and Winternitz One-Time Signatures (WOTS). Through an intuitive Tkinter GUI, the developed system enables users to sign and validate multiple messages, making it a useful and instructive tool for learning about post-quantum digital signatures.

Traditional cryptographic algorithms like RSA and ECC are seriously threatened by the emergence of quantum computing, which calls for the creation of post-quantum cryptography solutions. The SPHINCS+ hash-based signature scheme, a top contender for post-quantum digital signatures, is simplified in this project. By combining FORS (Forest of Random Subsets) and Winternitz One-Time Signatures (WOTS), the implementation offers a strong

INTRODUCTION:

Unprecedented computational power brought about by quantum computing can crack popular encryption schemes used to protect contemporary digital communication. Public-key cryptosystems based on factorization or discrete logarithms are susceptible to quantum attacks, as shown by algorithms like Shor's and Grover's. Consequently, the development of quantum-resistant algorithms through post-quantum cryptography has emerged as a crucial field of study. SPHINCS+ is a stateless hash-based signature scheme that is resistant to both classical and quantum attacks because it solely depends on the security of cryptographic hash functions. In order to inform users about hash-based cryptography concepts and their potential for future-proofing digital signatures, this project incorporates essential elements of SPHINCS+.

OBJECTIVE:

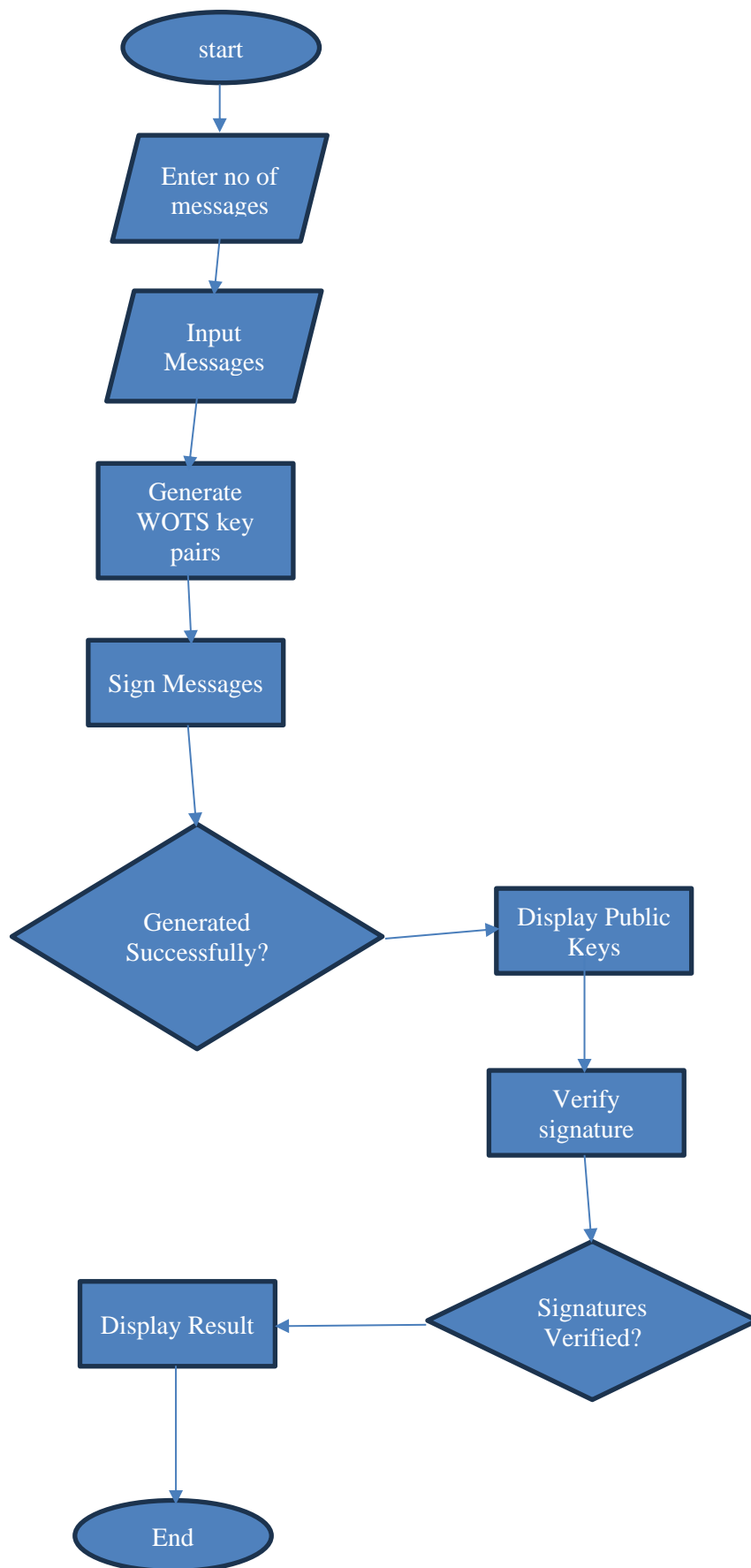
The main goal of the project is to illustrate the fundamental ideas behind hash-based digital signatures and highlight how important they are to protecting digital communications in the post-quantum era. The implementation demonstrates how to efficiently and securely sign and verify multiple messages by combining FORS (Forest of Random Subsets) and Winternitz One-Time Signatures (WOTS). This method not only sheds light on the algorithms' technical workings but also emphasizes how resilient they are to quantum attacks, which is a critical issue as quantum computing develops.

The project also makes use of an intuitive graphical user interface created with Tkinter, which streamlines communication and helps users comprehend difficult cryptographic ideas. The interface is a perfect teaching tool for researchers and students because it enables users to create signatures, input messages, and check authenticity step-by-step. Flexibility is ensured by the modular design, allowing for future improvements or integration with different cryptographic schemes.

The project intends to close the gap between theoretical cryptographic algorithms and their practical applications in addition to its educational value. It offers a useful illustration of how these signatures can guarantee data authenticity and integrity in delicate applications like blockchain technology, digital identity verification, and secure messaging. Additionally, the project lays the groundwork for future post-quantum cryptography research and development, opening the door for more sophisticated SPHINCS+ and related scheme implementations.

The project's ultimate goal is to increase awareness of how susceptible existing cryptographic systems are to quantum threats and encourage the use of quantum-safe substitutes. It promotes proactive measures to create a safe digital ecosystem that can withstand upcoming quantum challenges by demonstrating the usefulness and effectiveness of hash-based signatures.

FLOW CHART:



METHODOLOGY:

Create Keypairs for WOTS:

Utilize a random secret key (os. random) and use SHA-256 to hash the secret key to obtain a public key.

Signing Messages:

Determine the message's hash.

To create a signature, concatenate the message hash and the secret key.

Verification of Message:

Recalculate the hash of the message.

Compare it to the signature's hash section.

Integration of FORS:

To sign individual messages, use different WOTS keypairs.

For multi-message verification, aggregate the results.

Interface for Users:

Create an application using Tkinter to communicate with users.

Using buttons and fields, enable message input, signing, and verification.

MODULES:

Keypair Generation Module:

Generates and stores secret-public key pairs for WOTS.

Signing Module:

Takes input messages and generates cryptographic signatures.

Verification Module:

Verifies the authenticity of signed messages using public keys.

Graphical User Interface (GUI) Module:

Facilitates user interaction for key generation, signing, and verification.

Error Handling Module:

Handles user input errors and cryptographic failures gracefully.

BACKGROUND STUDY:

Traditional cryptographic systems' security is now in jeopardy due to the quick development of quantum computing. Quantum algorithms like Shor's can be used to attack algorithms like RSA and ECC, which depend on the difficulty of mathematical problems like discrete logarithms and integer factorization. Post-quantum cryptography solutions that are resistant to both classical and quantum attacks are now necessary as a result of this. Because hash-based cryptography relies on the security of cryptographic hash functions, which are secure even in the context of quantum computing, it has become one of the most promising options among the different strategies being investigated.

Hash-based signature schemes have drawn a lot of interest, including the eXtended Merkle Signature Scheme (XMSS) and its stateless successor SPHINCS+. To produce safe and effective digital signatures, these schemes make use of one-way hash functions. Specifically, SPHINCS+ is made to offer a scalable and stateless solution, which makes it appropriate for practical uses. It combines two fundamental components: FORS (Forest of Random Subsets), which allows batch signing of multiple messages, and Winternitz One-Time Signatures (WOTS), which efficiently sign individual messages.

The initiative also advances a wider knowledge of cryptographic hash functions, which are essential to digital signatures and a number of other applications, including blockchain technology, data integrity checking, and safe password storage. The project emphasizes the significance of implementing quantum-resistant systems as part of future-proof digital security strategies by fusing these theoretical underpinnings with real-world demonstrations, bridging the gap between academic research and its practical applications.

IMPLEMENTATION:

```
import os
import hashlib
import tkinter as tk
from tkinter import messagebox

def generate_wots_keypair():
    secret_key = os.urandom(32)
    public_key = hashlib.sha256(secret_key).digest()
    return secret_key, public_key

def wots_sign(secret_key, message):
    message_hash = hashlib.sha256(message.encode()).digest()
    return secret_key + message_hash

def wots_verify(public_key, signature, message):
    message_hash = hashlib.sha256(message.encode()).digest()
    return signature[-32:] == message_hash

class FORS:
    def __init__(self, num_signatures):
        self.num_signatures = num_signatures
        self.fors_keys = [generate_wots_keypair() for _ in range(num_signatures)]

    def sign(self, messages):
        signatures = []
        for i in range(self.num_signatures):
            signatures.append(wots_sign(self.fors_keys[i][0], messages[i]))
        return signatures

    def verify(self, public_keys, signatures, messages):
        for i in range(self.num_signatures):
            if not wots_verify(public_keys[i], signatures[i], messages[i]):
                return False
        return True

class SignatureApp:
```

```
    def __init__(self, master):
```

```

self.master = master
master.title("SPHINCS+ Hash-Based Signature Scheme")

self.label = tk.Label(master, text="Enter number of messages to sign:")
self.label.pack()

self.num_messages_entry = tk.Entry(master)
self.num_messages_entry.pack()

self.messages_frame = tk.Frame(master)
self.messages_frame.pack()

self.create_fields_button = tk.Button(master, text="Create Message Fields",
command=self.create_message_fields)
self.create_fields_button.pack()

self.sign_button = tk.Button(master, text="Generate Signatures",
command=self.generate_signatures)
self.sign_button.pack()

self.verify_button = tk.Button(master, text="Verify Signatures",
command=self.verify_signatures)
self.verify_button.pack()

def create_message_fields(self):
    try:

        num_messages = int(self.num_messages_entry.get())
        if num_messages <= 0:
            messagebox.showerror("Error", "Please enter a positive number of messages.")
            return

        self.messages = []
        self.signatures = []
        self.public_keys = []
        self.messages_frame.pack_forget()
        self.messages_frame = tk.Frame(self.master)

```

```
self.messages_frame.pack()
```

```
for i in range(num_messages):  
    message_entry = tk.Entry(self.messages_frame)  
    message_entry.pack()  
    self.messages.append(message_entry)
```

```
messagebox.showinfo("Instructions", "Enter your messages in the fields  
provided.")
```

```
except ValueError:  
    messagebox.showerror("Error", "Please enter a valid number.")
```

```
def generate_signatures(self):  
    try:
```

```
        messages_text = [entry.get() for entry in self.messages]  
        if any(msg == "" for msg in messages_text):  
            messagebox.showerror("Error", "All message fields must be filled.")  
        return
```

```
        self.fors_instance = FORS(num_signatures=len(messages_text))  
        self.signatures = self.fors_instance.sign(messages_text)  
        self.public_keys = [keypair[1] for keypair in self.fors_instance.fors_keys]
```

```
        messagebox.showinfo("Success", "Messages signed successfully!")
```

```
except Exception as e:  
    messagebox.showerror("Error", f"An error occurred: {e}")
```

```
def verify_signatures(self):  
    if not hasattr(self, 'signatures') or not hasattr(self, 'public_keys'):  
        messagebox.showwarning("Warning", "Please sign messages first.")  
    return
```

```
messages_text = [entry.get() for entry in self.messages]
verification_result = self.fors_instance.verify(self.public_keys, self.signatures,
messages_text)

if verification_result:
    messagebox.showinfo("Verification Result", "All signatures verified
successfully!")
else:
    messagebox.showerror("Verification Result", "Signature verification failed.")

if __name__ == "__main__":
    root = tk.Tk()
    app = SignatureApp(root)
    root.mainloop()
```

OUTPUT :

Enter number of messages to sign:

Create Message Fields

Generate Signatures

Verify Signatures

Enter number of messages to sign:

Create Message Fields

Generate Signatures

Verify Signatures


Enter number of messages to sign:

Create Message Fields

Generate Signatures


Verify Signatures

Verification Result

 All signatures verified successfully!

OK

Verification Result

 Signature verification failed.

OK

FUTURE SCOPE:

The project can grow in a number of ways. Real-world deployment in secure communication systems might be made possible by a complete SPHINCS+ implementation. In decentralized systems, integrating the application with blockchain networks may improve transaction security. Additionally, the tool can be improved for use as a teaching tool for training courses and workshops on cryptography. To improve the state of quantum-resistant systems, future studies can investigate hybrid models that combine hash-based and lattice-based cryptographic techniques.

CONCLUSION:

This project uses a simplified version of SPHINCS+ to illustrate the possibilities of post-quantum cryptography. It closes the gap between research and real-world use cases by fusing theoretical underpinnings with useful applications. Quantum resilience is guaranteed by the application of hash-based techniques, and a wider audience can understand these ideas thanks to the graphical user interface. Projects like this will be essential to the shift to safe, future-proof cryptographic systems as quantum computing advances.

REFERENCES:

- [1] Bernstein, D. J., Hülsing, A., Kölbl, S., Rijneveld, J., & Schwabe, P. (2019). SPHINCS+: Practical Stateless Hash-Based Signatures. *National Institute of Standards and Technology (NIST)*. Retrieved from <https://sphincs.org>
- [2] Hülsing, A., Butin, D., Gazdag, S., Rijneveld, J., & Schneider, S. (2018). XMSS: eXtended Merkle Signature Scheme. *RFC 8391*. Retrieved from <https://datatracker.ietf.org/doc/rfc8391/>
- [3] Albrecht, M., Player, R., & Scott, S. (2015). On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3), 169–203. DOI: 10.1515/jmc-2015-0016.
- [4] National Institute of Standards and Technology (NIST). (2023). Post-Quantum Cryptography Standardization Process. Retrieved from <https://csrc.nist.gov/projects/post-quantum-cryptography>
- [5] Shor, P. W. (1994). Algorithms for quantum computation: Discrete logarithms and factoring. *Proceedings of the 35th Annual Symposium on Foundations of Computer Science*, 124–134. DOI: 10.1109/SFCS.1994.365700
- [6] Grover, L. K. (1996). A fast quantum mechanical algorithm for database search. *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, 212–219. DOI: 10.1145/237814.237866
- [7] Buchmann, J., Dahmen, E., & Hülsing, A. (2011). XMSS - A Practical Forward Secure Signature Scheme Based on Minimal Security Assumptions. *Post-Quantum Cryptography PQCrypto 2011*. DOI: 10.1007/978-3-642-25405-5_6
- [8] Goldreich, O., Goldwasser, S., & Micali, S. (1988). How to construct random functions. *Journal of the ACM*, 33(4), 792–807. DOI: 10.1145/42282.42283
- [9] Lamport, L. (1979). Constructing Digital Signatures from a One-Way Function. *Technical Report SRI-CSL-98*. Retrieved from <https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/lamport79.pdf>
- [10] Boneh, D., & Shoup, V. (2020). *A Graduate Course in Applied Cryptography*. Version 0.5. Retrieved from <https://toc.cryptobook.us/>

