Microcontroller-based Implementation of ParseKey+ for Limited Resources Embedded Applications

Reza Makvandi

Submitted to the
Institute of Graduate Studies and Research
in partial fulfillment of the requirements for the Degree of

Master of Science
in
Electrical and Electronic Engineering

Eastern Mediterranean University
February 2011
Gazimağusa, North Cyprus

Approval of the Institute of Graduate Studies and Research

Prof. Dr. Elvan Yılmaz
Director (a)

I certify that this thesis satisfies the requirements as a thesis for the degree of Master of Science in Electrical and Electronic Engineering.

Assoc. Prof. Dr. Aykut Hocanın
Chair, Department of Electrical and Electronic Engineering

We certify that we have read this thesis and that in our opinion it is fully adequate in scope and quality as a thesis for the degree of Master of Science in Electric and Electronic Engineering.

Asst. Prof. Dr. Behnam Rahnama Co-Supervisor	Prof. Dr. Şener Uysal Supervisor

Examining Committee

1. Prof. Dr. Şener Uysal
2. Prof. Dr. Erden Başar
3. Assoc. Prof. Dr. Mustafa K. Uyguroğlu
4. Assoc. Prof. Dr. Rashad Aliyev
5. Asst. Prof. Dr. Behnam Rahnama

ABSTRACT

ParseKey+ is an approach to a new highly secure and safe authentication service. The scheme provides secure authentication process for both client and server sides. It employs hash to hide the encrypted key in a key file retrievable only by the other side knowing the indices and lengths of sub-keys hidden inside ParseKey+ file. The ParseKey+ file itself is also encrypted by the password of the other party in authentication service using a symmetric encryption method. The key file is recreated at each sign-on procedure; it provides an additional security layer beyond using the login password. ParseKey+ avoids client and server impersonations while guaranteeing mutual client/server authentication. Likewise, the ParseKey+ scheme avoids replay, meet-in-the-middle, ciphertext-only, and side-channel attacks. ParseKey+ relies on scattering sub-keys in a block of uniformly created random noise.

We wish to implement the ParseKey+ authentication system with limited resources on AVR microcontrollers. More sophisticated versions of these families of microcontrollers are widely used in PDAs, new generation of mobile phones and various embedded devices. Implementation of ParseKey+ allows securing Internet communication and transactions on above-mentioned devices in addition to providing a highly secure mechanism for implementing very low-cost hardware USB keys for online banking applications.

Keywords: Microcontroller-based Authentication, Embedded System, ParseKey+

ÖZ

ParseKey+ yeni, son derece güvenli bir kimlik doğrulama hizmet yaklaşımıdır. Bu yaklaşım hem istemci hem de sunucu tarafı için güvenli kimlik doğrulama işlemi sağlar. Hash kullanarak ParseKey+ dosyası içinde kilitlenmiş olan kodlanmış anahtarı sadece öteki taraftan endekseleri ve altanahtar uzunlukları bilenler tarafından alınabilir. ParseKey+ dosyasının kendisi de öteki partinin şifresi ile simetrik olarak kodlanmıştır. Her giriş işleminde anahtar dosya çağrılır; giriş kodu (password) kullanımına ilaveten artı bir güvenlik seviyesi sağlar. Karşılıklı istemci / sunucu kimlik doğrulaması garanti ederken ParseKey + istemci ve sunucu taklitleri önler. Aynı şekilde, ParseKey + düzeni tekrarı, ortada buluşmayı, şifreli-sadece ve yan kanal saldırılarını önler. ParseKey + düzgün oluşturulan rastgele gürültü bir blok saçılma alt anahtarlarına dayanmaktadır.

Biz, sınırlı kaynaklara sahip AVR mikroişlemcileri üzerinde ParseKey + kimlik doğrulama sistemi uygulamak istiyoruz. Mikroişlemcisi daha gelişmiş versiyonları yaygın PDA'lar, çeşitli gömülü cihazlar ve yeni nesil cep telefonlarında kullanılmaktadır. ParseKey+ Uygulama online bankacılık uygulamaları için çok düşük maliyetli donanım USB anahtarları uygulanması için son derece güvenli bir mekanizma vermenin yanı sıra yukarıda bahsedilen cihazlarda internet iletişimi ve işlemlerinde güvenli olaral kullanılmaktadır.

Anahtar Kelimeler: Mikrodenetleyici tabanlı Doğrulama, Gömülü Sistem, ParseKey+

I wish to dedicate this thesis to my lovely wife (Sara) and my dear parents (Mrs. Aghdas Abaszade and Mr. Bahman Makvandi) who have always supported and encouraged me.

ACKNOWLEDGMENTS

I take this opportunity to thank my supervisors Prof. Dr. Şener Uysal and Asst. Prof. Dr. Behnam Rahnama for guiding me all along my thesis research. Whom without their continuous help, I believe, I could not reach to this point.

I would like to express my sincere appreciation to the jury committee members, Prof. Dr. Erden Başar, Assoc. Prof. Dr. Mustafa K. Uyguroglu, Assoc. Prof. Dr. Rashad Aliyev and Assoc. Prof. Dr. Hasan Demirel, who accepted to share their time in reviewing my research and then directing me at the thesis defense meeting.

I also wish to thank all the faculty members at the department of Electrical and Electronic Engineering, and specially the chairman, Assoc. Prof. Dr. Aykut Hocanın, who have given me the great opportunity of research assistantship.

In addition, I wish to sincerely thank all my friends who kindly shared their knowledge and experiences with me.

Last but not least, I would like to express my deepest appreciation to my family who allowed me to travel to Cyprus and supported me during my study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

AES	Advanced Encryption Standard
DES	Digital Encryption Standard
HMAC	Hash Message Authentication Code
HTTP	Hypertext Transfer Protocol
ITU-T	International Telecommunication Union – Telecommunication Standardization Sector
MAC	Message Authentication Code
MD5	Message Digest 5
NIST	National Institute of Standards and Technology
NSA	National Security Agency
OTP	One-Time Password
SD	Secure Digital
RSA	Rivest-Shamir-Adleman, a public-key crypto algorithm
SHA	Secure Hash Algorithm
SHS	Secure Hash Signature Standard
URI	Unified Resource Identifier
WPA2	Wi-Fi Protected Access 2

Chapter 1

INTRODUCTION

Authentication is the process of verifying the claimed identity of a client. The network elements (NE) environment must offer features to confirm the claimed identity of a client before giving that client operations access. Depending on the NE and the applications, there could be different kinds of authenticators. Recently PITAC (President's Information Technology Advisory Committee) while looking into cyber security issues pointed out the importance of authentication service [1]. They listed authentication research as the top major among a list of 10 major areas for the current decade. In fact applications were distributed and carefully examined through connectionless services where authentication of parties has gained highly importance. Whereas protecting servers against illegitimate users has been a priority of the industry, security tools must be used also to protect users as fake website developers become more innovative.

Cipher analysis of authentication services through not safe networks shows possibility of misappropriation of protected information [2]. Such systems must include robust security mechanisms to guarantee the safety of private information [3], large amounts of operational and process data, such as ontology files in deep Web.

An authentication service provides safe access to protected information on unsafe networks and systems by controlling and managing interaction [4]. It checks login data such as username, password, and biometric information, and, permits to login such cases where user-supplied data matches server-side data [5].

In order to increase the security of an authentication service, additional security steps besides exchanging username/password are needed. ParseKey+, the proposed authentication service, uses transposition of scattered sub-keys in uniform random noise for key encryption and then encrypts the key file itself using a symmetric encryption method, safely authenticates legitimate clients and servers to each other. ParseKey+ is improved by using transposition of scattered sub-keys in uniform random noise spread over a ParseKey+ file.

1.2. Network Security

Information is the core of today's business. All organizations possess critical or sensitive information. First, information security ensures that protective tools are properly implemented. Second, information security is intended to protect the information. Information must be protected because it has value, and that value comes from the characteristics of the information.

Network security is the protection of networking competent, connections and contents. The concept often changing with different "perspective" varies according to [6]. For example: users (individuals, businesses, etc.) point of view, their demands for personal privacy or commercial interests of the information in the transmission network is subject to confidentiality, integrity and authenticity of protection, avoid other people or opponent use of wire-tapping, impersonation, tampering with, or deny, such as means of violating the interests of users and hidden, but also prevent other users of non-authorized access and destruction. For the operation of network and managers, they want information on the local network access, read and write operations such as protection and control, to avoid the "trapdoor", viruses, unauthorized access, denial of service and network resources illegal occupation and illegal control, and other threats, repression and defend the network from hackers.

From the security sector side, they hope that illegal, harmful or state secret filter the information and block it to avoid leaking confidential information to avoid harm to the community and the country resulting in tremendous losses. From a social perspective, education and ideology, unhealthy content on the network, will be the stability of society and human development that are causing obstruction must be controlled.

In an essence, network security is the security of information networks, refers to the network system hardware, software and system data will be protected from accidental or malicious destruction and the reasons for the change, disclosure, continuous and reliable system normal operations, network services without disruption. Broadly speaking, any information relating to the network confidentiality, integrity, availability, authenticity and controllability of the relevant technical and theoretical network security are on the field.

The content of network security involves both technical aspects of the problem as well as management issues. The two of them complement each other, and neither is in dispensable. Focused primarily on technical aspects to prevent unauthorized users external attacks, the management will focus on the internal human factors management. Ways to more effectively protect critical information and data to enhance the safety of the computer network system has led to a scenario where all applications must be considered and the need to address them an important issue.

In the different environment, network security also generates different issues. They are addressed as the following:

• Operating system security: that is information processing and transmission system security. It focuses on ensuring system uptime, and avoids the damage and the collapse of the system storage, avoids information getting lost and damaged during processing and transmission of information, and avoids electromagnetic leakage, have leaked information, interfering with other, the interference of others.

• Network information security system: including user password authentication, user access control, data access, control, security, auditing, security tracking, the computer virus prevention and data encryption.

• Dissemination of information on the network security: the consequences of the dissemination of information security. Including information filtering, it focuses on the prevention and control of illegal and harmful information dissemination of the consequences. Avoid large public network to transmit information out of control.

• Information on the content of network security: It focuses on the protection of confidentiality of information, authenticity and integrity. Avoid the use of system attacks the security flaw eavesdropping, impersonation, fraud, and undermine the legitimate user behavior. In essence, is to protect the interests of users and privacy.

All the techniques for providing security have two components according to [6]:

1. A security-related transformation on the information to be sent. Examples include the encryption of the message, which scrambles the message so that it is unreadable by the opponent, and the addition of the code based on the contents of the message, which can be used to verify the identity of the sender.

2. Some secret information shared by the two principals and remains unknown to the opponent. An example is an encryption key used in conjunction with the transformation to scramble the message before transmission and unscramble it on reception.

1.3 Authentication

Authentication is the process of verifying the claimed identity of a user. The network elements (NE) environment must offer features to verify the claimed identity of a user before giving that user operations access. Depending on the NE and the applications, there could be different kinds of authenticators. For example according to [7]:

• The user can be associated with confidential information that only he or she is supposed to possess such as: password, private key, or randomly time-varying PIN (such as those provided by single-use password tokens).

• The user can be associated with a distinctive physical or logical address (e.g., user's authorized directory number, network address).

• The user can be authenticated by certain unique attributes such as: voice or speech pattern, handwriting style, palm print, or retina scan.

There are three fundamental techniques used in authentication mechanisms [8]:

• Something you know, which usually refers to passwords and personal identification numbers (PINs). The simplest implementations of passwords and PINs yield the simplest of all authentication mechanisms.

• Something you have, which usually refers to cards or tokens. Physical authentication devices, such as smart cards and password tokens, were developed to eliminate certain weakness associated with passwords. A major benefit of cards and tokens is that they can't be shared with the same freedom as sharing passwords.

• Something you are, which refers to biometrics- the measurement of physical characteristics or personal traits. Common biometric verification techniques try to match measurements from one user's fingerprint, hand, eyes, face, or voice to measurements that were previously collected from him/her.

There are two general applications for this: identification and verification. "With identification, the biometric system asks and attempts to answer the question, 'who is X?' Verification occurs when the biometric system asks and attempts to answer the question, 'is this X?' after the user claims to be X.

In a verification application, the biometric system requires input from the user, at which time the user claims his or her identity via a password, token, or user name. This user input points the system to a template in the database. The system also requires a biometric sample from the user. It then processes and compares the sample to or against the user. This is called a "one to-one search (1:1)" [8].

1.3.1 Authentication Protocols and Methods

This section gives the reader a general understanding that there exist multiple methods of authentication, in which some are more secure than others. There are some de facto standards in authentication such as basic authentication i.e. using a user inputted username and password combination. In this section, different authentication methods are described in order of complexity and strength. The most interesting currently widely used authentication method addressed in this research is the public-key certificate based authentication scheme described as following.

1.3.1.1 User ID / Password Authentication

Historically, the use of plain username and password combinations has been the most common method of authentication. Today, it is still the most prevalent authentication method in existence, and its demise is nowhere in sight. It is an outdated, insecure method, but easy to implement with minimal requirements on the user-terminals with regard to equipment and software. Due to this and large legacy payload, it will still be in use well into the 21st century.

1.3.1.2. Basic Authentication

Basic authentication means a plaintext username/password pair, which is inputted into a dialogue box, a form or prompted for this information. Then the textual information is checked against a database of correct answers, which one stores in plaintext or in some hashed form in a text file or in a database within the authorization system.

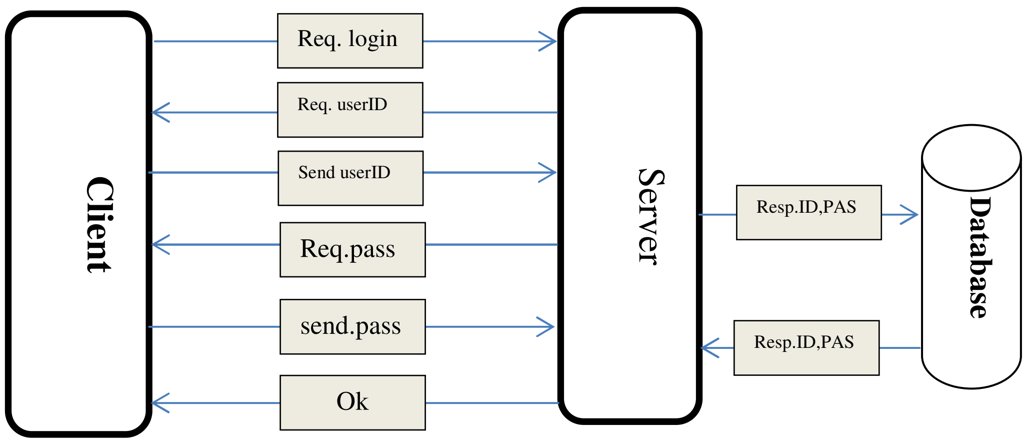


Figure 1.1: simplest occurrence of basic authentication

In “Figure 1.1, the simplest way of authenticating a user ID and a password is depicted. Here the server asks the user to identify him and after the user has supplied identification information, his user ID, the server then prompts for his password and checks if it exists on the system, and if the password matches the one stored in its internal password database. If there is a match, the user is granted access to the server.

In the web environment the HTTP protocol specifies a challenge-response framework for a web server to request authentication credentials from a client by sending the client a “401 Unauthorized” error message. The user-agent may respond to this challenge with an Authorization-header field with the http-request [9]. This information is then sent unencrypted over the network to the service for authentication. If the given credentials are correct, the client is given access i.e. authenticated and authorized to access the service. When the identity of the client is established access control decisions can be made as dictated by the access policy of the service.

1.3.1.4 Digest Authentication

Similar to basic authentication, digest access authentication verifies that both communicating parties share a secret (a password); unlike basic authentication, this verification can be done without sending the password unscrambled, which is the biggest drawback of basic authentication [9].

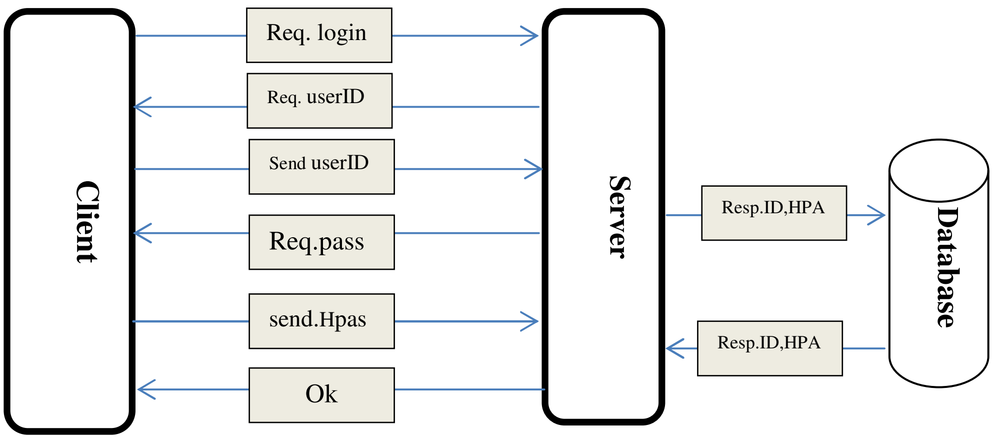


Figure 1.2: Simplified Digest Authentications

Digest authentication operates much the same way as basic authentication as is illustrated in Figure 1.2, but with the slight modification that only hashes of the password are transported over the Internet instead of a plaintext password. The digest scheme issues challenges using a nonce value. A valid response contains a checksum of the username, password, a given nonce value, the HTTP method and the requested URI. This way, the password is never transmitted unscrambled over the Internet [9]. The basic authentication scheme is not considered a secure method of user authentication since the user name and password are passed over the network as plaintext.

1.3.1.5 One-Time Password

One-time password (OTP) is a special case of basic authentication where the password changes every time one authenticates to a service and none of the passwords is reusable. When processing OTP-authentication the server retains the correct passwords in a secure index so that when one authenticates, only the next unused password is valid. This protects the authentication process from replay attacks in which an eavesdropper has recorded previous network traffic and discovered the username/password pair and attempts to log into the protected service using these credentials. There are two entities in the operation of the OTP one-time password system. The generator must produce the appropriate one-time password from the user's secret pass-phrase and from information provided in the challenge from the server.

The server must send a challenge, which includes the appropriate generation parameters to the generator. The one-time password must be verified, stored and correspond to the sequence number [10].

This requires that the server does not contain any compromising secret information while the seed, sequence number and last used key are all public data and non-compromising given that the secure hash function used to generate the password-sequence is non-invertible.

The OTP system generator passes the user's secret pass-phrase, along with a seed received from the server as part of the challenge, through multiple iterations of a secure hash function, which produces a one-time password. After each successful authentication, the number of secure hash function iterations is reduced by one, which generates a sequence of unique passwords. The server verifies the one-time password received from the generator by computing the secure hash function once and comparing the result with the previously accepted one-time password [10].

The generator on the other hand must be reliable and secure as it contains the secret generation key with which it computes the required number of hashes to generate the correct password.

In the OTP system the password is coded as six human readable words as shown in Table 2.1 to encode the 64 bit long password into a more easily typed version [10] the standard dictionary for this encoding is documented in the S/Key RFC 1760 [5].

There are a few variants of OTP like the time-based variant of SecurID™ from RSA Security Inc., which is a hardware token that generates new passwords every 60 seconds that are valid only for that period of time. In the SecurID™ scheme the token and the authentication server are clock-synchronized and seeded with the same secret start value [11]. Once the system has been activated, the stream of "random" numbers the token generates is identical on all similarly primed tokens. Usually, this random number is appended to a static personal secret for added security, so that the possession of the token in itself does not permit access to protected resources.

This research tries to implement ParseKey+ on limited resource AVR series microcontrollers. In order to accomplish this mission, firstly, clients need to work with familiar device (USB Connection) to connect to (for example) their ATM station. In addition, ParseKey+ should work with encryption protocol like AES and hash function (SHA) and stores data in database, also this application requires communicating with server via specific application for an operating system such as Windows or Mac OS. To achieve this goal, TEENSY2 was chosen as hardware (details were explained in chapter 4).

Secondly, C Programming Language on AVR Studio compiler was chosen for writing the required microcontroller programs. Finally, we aim to provide a platform for making highly secure mechanism for implementing hardware USB keys for online banking applications.

Explanation of encryption and hashing functions and discussing about available solution are introduced in chapter 2. In chapter 3, the new approach is presented. We also detail the limitations and the existing problems.

In chapter 4, practical experiment and results and comparison with existing solutions are presented. Chapter 5 covers a short conclusion and we discuss possible future works.

Chapter 2

IMPERSONATION AVOIDANCE PROCEDURE

This chapter introduces the necessity of designing secure environment in network communication by choosing a policy to process verifying of the claimed identity of users. In addition, cryptography, encryption service and hashing functions are explained in detail. Furthermore, it gives a brief explanation on steganography as it is necessary for the explanation of the future work research. We wish to also consider available solutions and their vulnerabilities so that the need for securer methods founds to be vital.

2.1 Cryptography

Cryptography is the process of translating data into a scrambled code that can be decrypted and sent across a public or private network. Cryptography plays a central role in modern authentication and single sign-on systems. It is used in various ways from generating certificates, signatures, protecting data and traffic to storing credentials locally in a secure manner. The most common cryptographic transformations are briefly explained as follows.

2.1.1 Hashing and Message Authentication

The hash functions are used for generating unique "fingerprints" of an arbitrary amount of data. This fingerprint is then utilized in digital signatures as the representative of the actual document that is signed. These fingerprints are also used for integrity checking purposes in the message authentication form.

Hashing is the process of using a one-way mathematical function on a message M to render it into a unique fixed-size hash code. The hash code is a function of every single bit of M, H (M), and therefore if any bit in M changes, the correct hash code will change [12] [13].

Mathematically the significant property of hash-functions is collision resistance [12] [13]. This means that if a hash-function is collision free and no two different objects can hash to an identical hash-value. Because of which the hash of a message M, H (M) can act as a "fingerprint" -like unique identifying value of the message M. Hashes are typically used for storing passwords, message authentication and verifying the signatures of electronic documents. The three typical hash algorithms used are Secure Hash Algorithm (SHA), Message Digest 5 (MD5) [12], [13].

A hash code can be used to provide message authentication i.e. integrity checking by appending the hash code as redundant data to the message M. Using a plain hash of the data is in itself insufficient, and therefore, a secret component needs to be added to the hashed message. These hash-functions are called keyed-hashes or message authentication functions [12], [13].

This differs from conventional hash-functions in that a secret key 'K' is appended to the message M so that MAC = H (M, K) and an attacker cannot re-compute the hash from a modified message M' simply by MAC' = H (M'). Because he does not know the secret key K, this is impossible. MAC algorithms can be built based on conventional hash-functions or symmetric encryption algorithms with modifications to make them one-way. Popular MAC algorithms are HMAC [12], [13] and the Data Authentication Algorithm [13].

2.1.2 Symmetric Cryptography

With symmetric cryptography, the cryptographic function uses the same key for encryption and decryption. Symmetric cryptography is sometimes called shared-secret or secret key cryptography because the encryption/decryption key has to be shared between all parties authorized to access the enciphered data. Symmetric cryptography is used for the encryption of bulk traffic because it is less demanding on computer resources than public-key based cryptography [12].

The downside of symmetric cryptography is the problem of key management, [13] and the trustworthiness of those involved with the shared secret key. William Stallings [12] remarks that public-key cryptography also requires a set of protocols for key distribution, and therefore it is not a panacea for key distribution problems.

The most common symmetric algorithms in use today are the Data Encryption Standard (DES) and 3DES – a variant of DES with a triple length encryption key. More recent symmetric encryption algorithms include IDEA, Blowfish, Two Fish, Rijndael, etc. Rijndael won the contest to become the Advanced Encryption Standard [16], for the US National Institute of Standards and Technology. All of these newer algorithms use longer encryption keys than DES, which makes them much harder to compromise if no vulnerabilities in the algorithms themselves are uncovered.

2.1.3 Asymmetric Cryptography

Asymmetric cryptography, also known as public-key cryptography, is fundamentally different from symmetric cryptography. The difference between these two is in the level of secrecy of the encryption keys. In symmetric cryptography, it is of primary concern to keep the encryption key secret. In asymmetric cryptography, one has two different keys, the public key, which can be revealed to anyone and the secret key that is personal and must be kept secret [12].

The main difference is that anyone can now send you securely encrypted material, which can be decrypted only with your private key – the public key is not able to decrypt the encrypted data – only encrypt it.

These two keys are connected to each other mathematically in such a way that one cannot deduce the other without knowledge of the original generation prime numbers used to create the key-pair. The RSA algorithm is the most popular and widely used public-key algorithm on the market, named after its inventors Ron Rivest, Adi Shamir and Leonard Adleman [13].

2.1.4 Public-Key Digital Signatures

The digital signature algorithm, DSA, was developed by the NSA and became the digital signature standard, DSS, of NIST in 1994. One of the main reasons DSA was chosen over the de facto standard RSA was the royalty freeness of DSA vs. the RSA patents and royalty requirements [13]. It was originally thought possible to generate only digital signatures and therefore be exempted from export restrictions. This assumption was shown to be in some cases false as demonstrated in the book Applied Cryptography. A digital signature is made by encrypting the hash of the document to be signed with the signer’s secret key and by appending the resulting value to the document. This signature can be verified by calculating the same hash-function of the document and comparing this with the signed hash-value after decrypting it with the signer's public-key.

The RSA algorithm can also be used for digital signatures and nowadays it may be used without royalties, so there are not many reasons left to use DSA anymore as its signature verification is slower than RSA's [13].

2.1.5 Key Exchange Algorithms

With key exchange algorithms, one can negotiate a symmetric encryption key for bulk data encryption in such a way that an eavesdropper cannot deduce the key from public information transmitted over the network in the key exchange negotiation. A very successful key exchange algorithm is the Diffie-Hellman key exchange algorithm [12]. There also exist algorithms that implement this with public-key encryption utilizing certificates [14].

2.1.6 AES (Advanced Encryption Standard)

AES is the abbreviation of Advanced Encryption Standard and is a United States encryption standard defined in Federal Information Processing Standard (FIPS) 192, published in November 2001. It was ratified as a federal standard in May 2002. AES is the most recent of the four current algorithms approved for federal us in the United States. One should not compare AES with RSA, another standard algorithm, as RSA is a different category of algorithm. Bulk encryption of information itself is seldom performed with RSA.RSA is used to transfer other encryption keys for use by AES for example, and for digital signatures. AES is a symmetric encryption algorithm processing data in block of 128 bits. A bit can take the values zero and one, in effect a binary digit with two possible values as opposed to decimal digits, which can take one of 10 values. Under the influence of a key, a 128-bit block is encrypted by transforming it in a unique way into a new block of the same size [15]. AES is symmetric since the same key is used for encryption and the reverse transformation, decryption. The only secret necessary to keep for security is the key. AES may configured to use different key-lengths, the standard defines 3 lengths and the resulting algorithms are named AES-128, AES-192 and AES-256 respectively to indicate the length in bits of the key [16] .

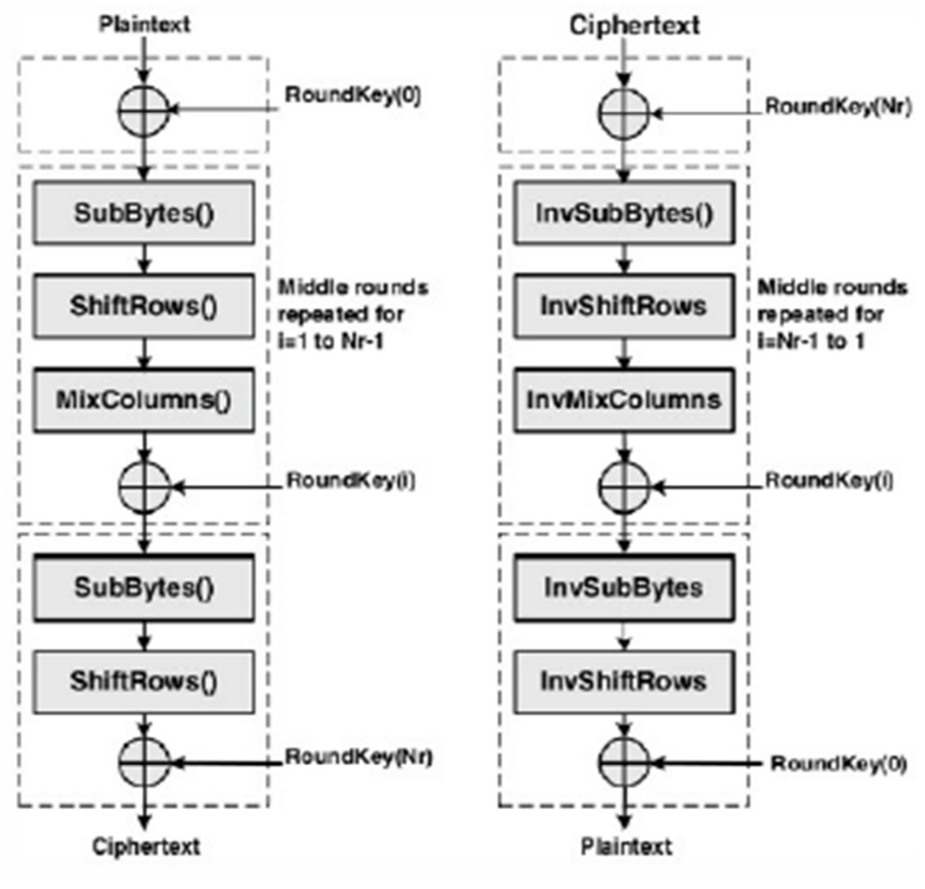


Figure 2.1: the AES encryption and decryption process [16].

Each additional bit in the key effectively doubles the strength of the algorithm, when defined as the time necessary for an attacker to stage a brute force attack, i.e. an exhaustive search of all possible key combinations in order to find the right one.

2.1.6.1 History of AES

In 1997 the US National Institute of Standards and Technology put out a call for candidates for a replacement for the ageing Data Encryption Standard, DES. 15 candidates were accepted for further consideration, and after a fully public process and three open international conferences, the number of candidates was reduced to five. In February 2001, the final candidate was announced and comments were solicited. 21 organizations and individuals submitted comments. None had any reservations about the suggested algorithm [15].

AES is founded on solid and well-published mathematical ground, and appears to resist all known attacks well. There's a strong indication that in fact no back-door or known weakness exists since it has been published for a long time, has been the subject of intense scrutiny by researchers all over the world, and such enormous amounts of economic value and information is already successfully protected by AES. There are no unknown factors in its design, and it was developed by Belgian researchers in Belgium therefore voiding the conspiracy theories sometimes voiced concerning an encryption standard developed by a United States government agency. A strong encryption algorithm need only meet only single main criteria.

- There is no way to find the unencrypted clear text if the key is unknown, except brute force, i.e. to try all possible keys until the right one is found.

A secondary criterion must also be met:

- The number of possible keys must be large so that it is computationally infeasible to actually stage a successful brute force attack in short enough a time.

The older standard, DES or Data Encryption Standard, meets the first criterion, but no longer the secondary one – computer speeds have caught up with it, or soon will. AES meets both criteria in all of its variants: AES-128, AES-192 and AES-256.

AES may, as all algorithms, be used in different ways to perform encryption. Different methods are suitable for different situations. It is vital that the correct method is applied in the correct manner for each and every situation, or the result may well be insecure even if AES as such is secure. It is very easy to implement a system using AES as its encryption algorithm, but much more skill and experience is required to do it in the right way for a given situation [15].

2.1.6.2 Structure of AES

In this section the structure of AES has been explained briefly. The input to the encryption and decryption algorithm is a single 128-bit block; this block is depicted as a square matrix of bytes. This block is copied into the State array, which is modified at each stage of encryption or decryption. After the final stage, State is copied to an output matrix. Similarly, the 128-bit is depicted as a square matrix of bytes. This key is expanded into an array of key schedule words; each word is 4 bytes and the total key schedule is 44 words for the 128-bit key. Ordering of bytes within a matrix is by column. A 128-bit plaintext is treated as a byte matrix of size 4x4, where each byte represents a value in GF (2^8). An AES round applies four operations to the state matrix in figure 2.2

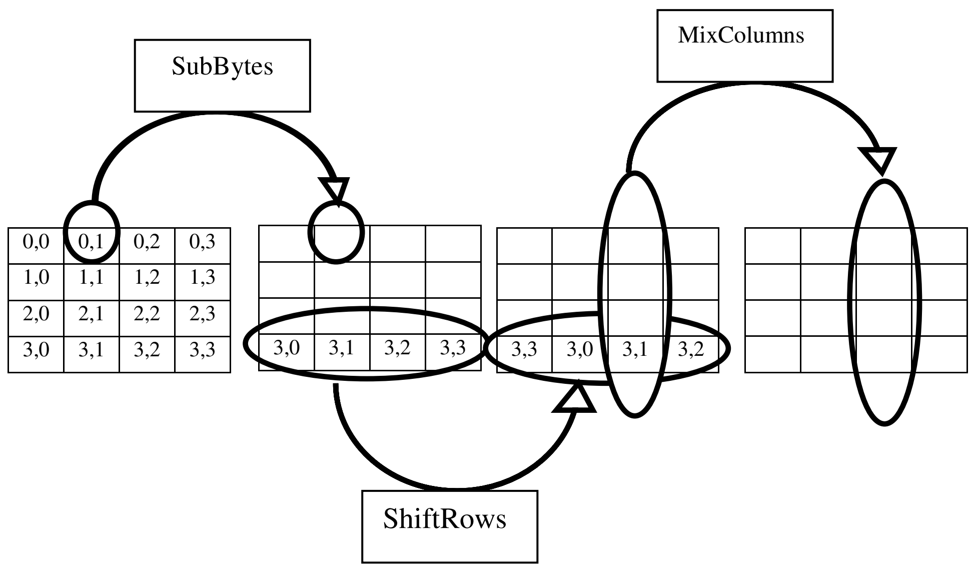


Figure 2.2: AES round

The SubBytes transformation is a non-linear byte substitution that acts on every Byte of the state in isolation to produce a new byte value using an S-box substitution table. This substitution, which is invertible, is constructed by composing two transformations:

1. First the multiplicative inverse in the finite field described earlier, with the $\{\mathbf{00}\}$ element mapped to itself.

2. Second the affine transformation over GF (2) defined by:

$$b_i = b_i \oplus b_{(i+4)mod8} \oplus b_{(i+5)mod8} \oplus b_{(i+6)mod8} \oplus b_{(i+7)mod8} \oplus c_i$$

For $0 \leq i < 8$ where b_i is bit i of the byte and c_i is bit i of a byte c with the value {63} or {01100011}. Here and elsewhere a prime on a variable on the left of an equation indicates that its value is to be updated with the value on the right.

The ShiftRows transformation operates individually on each of the last three rows of the state by cyclically shifting the bytes in the row such that:

$$S_{r,c} = S_{r,(c+h[r,Nb])modNb}\ for\ 0 \leq r < Nb\ and\ 0 < r < 4$$

The MixColumns transformation acts independently on every column of the state and treats each column as a four-term polynomial.

$$\begin{bmatrix} S'_{0,c} \\ S'_{1,c} \\ S'_{2,c} \\ S'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \\ S_{3,c} \end{bmatrix} for\ 0 \leq c < Nb$$

2.1.7 SHA (Secure Hash Algorithms)

In 1992, NIST announced a proposed standard for a collision-free hash function. The algorithm for producing the hash value is known as the Secure Hash Algorithm (SHA), and the standard using the algorithm in known as the Secure Hash Standard (SHS). Later, an announcement was made that a scientist at NSA had discovered a weakness in the original algorithm. A revision to this standard was then announced as FIPS 180-1, and includes a slight change to the algorithm that eliminates the weakness. This new algorithm is called SHA-1 [17].

This standard specifies four secure hash algorithms, SHA-1, SHA-256, SHA-384, and SHA- 512. All four of the algorithms are iterative, one -way hash functions that can process a message to produce a condensed representation called a message digest. These algorithms enable the determination of a message's integrity: any change to the message wills, with a very high probability, result in a different message digests. This property is useful in the generation and verification of digital signatures and message authentication codes, and in the generation of random numbers (bits). Each algorithm can be described in two stages: preprocessing and hash computation. Preprocessing involves padding a message, parsing the padded message into m-bit blocks, and setting initialization values to be used in the hash computation. The hash computation generates a message schedule from the padded message and uses that schedule, along with functions, constants, and word operations to iteratively generate a series of hash values. The final hash value generated by the hash computation is used to determine the message digest. The four algorithms differ most significantly in the number of bits of security that are provided for the data being hashed this is directly related to the message digest length. When a secure hash algorithm is used in conjunction with another algorithm, there may be requirements specified elsewhere that require the use of a secure hash algorithm with a certain number of bits of security [17].

For example, if a message is being signed with a digital signature algorithm that provides 128 bits of security, then that signature algorithm may require the use of a secure hash algorithm that also provides 128 bits of security (e.g., SHA-256). Additionally, the four algorithms differ in terms of the size of the blocks and words of data that are used during hashing [17].

2.1.7.1 Hash Function

A hash function H is a transformation that takes a variable-size input ‘m’ and returns a fixed- size string, which is called the hash value ‘h’ (that is, h = H(m)). Hash functions with just this property have a variety of general computational uses, but when employed in cryptography the hash functions are usually chosen to have some additional properties.

The basic requirements for a cryptographic hash function are,

- The input can be of any length,

- The output has a fixed length,
- H(x) is relatively easy to compute for any given x,
- H(x) is one-way,
- H(x) is collision-free.

A hash function H is said to be one-way if it is hard to invert, where "hard to invert" means that given a hash value ‘h’, it is computationally infeasible to find some input ‘x’ such that H(x) = h. If, given a message x, it is computationally infeasible to find a message y not equal to x such that H(x) = H(y) then H is said to be a weakly collision-free hash function. A strongly collision-free hash function H is one for which it is computationally infeasible to find any two messages x and y such that H(x) = H(y) [18].

The hash value represents concisely the longer message or document from which it was computed; one can think of a message digest as a "digital fingerprint" of the larger document. Examples of well-known hash functions are MD2 and MD5 and SHA. Perhaps the main role of a cryptographic hash function is in the provision of digital signatures. Since hash functions are generally faster than digital signature algorithms, it is typical to compute the digital signature to some document by computing the signature on the document's hash value, which is small compared to the document itself. Additionally, a digest can be made public without revealing the contents of the document from which it is derived. This is important in digital time stamping where, using hash functions, one can get a document time stamped without revealing its contents to the times tamping service [18].

2.1.8 Steganography

Steganography conceals the fact that a message is being sent. It is a method akin to covert channels, spread spectrum communication and invisible inks which adds another step in security. A message in cipher text may arouse suspicion while an invisible message will not.

Steganography is the art of concealing the existence of information within seemingly innocuous carriers. Steganography can be viewed as akin to cryptography. Both have been used throughout recorded history as means to protect information. At times these two technologies seem to converge while the objectives of the two differ. Cryptographic techniques "scramble" messages so if intercepted, the messages cannot be understood. Steganography, in an essence, "camouflages" a message to hide its existence and make it seem "invisible" thus concealing the fact that a message is being sent altogether. An encrypted message may draw suspicion while invisible messages will not [19].

There are a large number of steganographic methods that most of us are familiar with, ranging from invisible ink and microdots to secreting a hidden message in the second letter of each word of a large body of text and spread spectrum radio communication. With computers and networks, there are many other ways of hiding information, such as,

- Hidden text within Web pages.
- Hiding files in "plain sight".
- Null ciphers.

Steganography today, however, is significantly more sophisticated than the examples above suggest, allowing a user to hide large amounts of information within image and audio files. These forms of steganography often are used in conjunction with cryptography so that the information is doubly protected; first it is encrypted and then hidden so that an adversary has to first find the information (an often difficult task in and of itself) and then decrypt it.

There are a number of uses for steganography besides the mere novelty. One of the most widely used applications is for so-called digital watermarking. A watermark, historically, is the replication of an image, logo, or text on paper stock so that the source of the document can be at least partially authenticated. A digital watermark can accomplish the same function; a graphic artist, for example, might post sample images on her Web site complete with an embedded signature so that she can later prove her ownership in case others attempt to portray her work as their own.

Steganography can also be used to allow communication within an underground community. There are several reports, for example, of persecuted religious minorities using steganography to embed messages for the group within images that are posted to known Web sites [20].

2.2 Available Solutions

This section tries to introduce and describe about existing applications were used in authentication process in network communication. X.509, Lamport OTP, Yen-Shen-Hwang's One-Time Password and WPA2 are available solution were explained as follow.

2.2.1 X.509 Authentication Service

ITU-T recommendation X.509 is a part of the X.500 series of recommendations that define a directory service. The directory is a server or distributed set of servers that maintains a database of information about users. The information includes a mapping from user name to network address, as well as other attributes and information about the users.

X.509 defines a framework for the provision of authentication services by the X.500 directory to its users. The directory may serve as a repository of public-key certificates. Each certificate contains the public key of a user and is signed with the private key of a trusted certification authority. X.509 defines alternative authentication protocols based on the use of public-key certificates. X.509 is an important standard because the certificate structure and authentication protocols defined in X.509 are used in variety of contexts [21].

2.2.2 Lamport OTP

Password authentication is one of the simplest and the most comfortable authentication services. It provides the legitimate client to use the information in network. The user's password is stored in plaintext form in password table which is vulnerable to the revelation of the passwords. To solve this problem, Lamport proposed one-time password using one-way hash function [24]. Passwords can be compromised when they cross a network or from a server's database. In previous considerations, we assumed that there is a trusted third party and/or password database on the server side is safe. But this may not be true, and OTP schemas give solution for such cases.

Solution is strongly based on one-way hash functions. Lamport's schema allows having some finite number, N of authentications of a user to a server before the initialization procedure will be required. In initialization procedure, the user and server exchange securely secret information (by means of some special channel, personally, by ordered mail, courier, or in some other secure way). Schema assumes that a password never crosses insecure network, and the server's password database might be compromised, but can't be changed by an intruder. The server and user use one and the same hash function, h in the authentication procedure. The server authenticates the user applying the hash function to a current "password" value in its database. Actually, this value is derived from the passport, and is used as a current password, which changes from one authentication to another. That's why the schema is called "one-time password". Current password depends on the authentication number and can't exceed N. OTP schemas represent "challenge-response" schemas [21].

2.2.3 Yen-Shen-Hwang's One-Time Password

YEH-SHEN-HWANG's scheme is composed of registration stage, login stage and authentication stage. The scheme uses SEED as a pre-shared secret to resist against server spoofing attacks, replay attacks and off-line dictionary attacks. Also, it uses smart cards to securely preserve a pre-shared secret SEED and simplify the user login process. Moreover, it provides a session key to enable confidential communication over the network [21].

2.2.4 WPA2

WPA2is security method for wireless networks that provides stronger data protection and network access control. It provides enterprise and consumer Wi-Fi users with a high level of assurance that only authorized users can access their wireless networks. Based on the IEEE 802.11i standard, WPA2 provides government grade security by implementing the National Institute of Standards and Technology (NIST) FIPS 140-2 compliant AES encryption algorithm and 802.1x- based authentication.

There are two versions of WPA2: WPA2-Personal, and WPA2-Enterprise. WPA2-Personal protects unauthorized network access by utilizing a set-up password. WPA2-Enterprise verifies network users through a server. WPA2 is backward compatible with WPA [21].

2.3 Vulnerabilities of Available Approaches

Approaches defined above in authentication services require a hash function to transpose sub-keys. Moreover, the password itself is kept as a hash stream in database. However, in February 2005, an analysis to compromise the full sha1 by Xiao Yum Wang, Yigun Lisa Yin, and Hangbo Yu was reported (Wang, Yin, Yu, 2005). The attacks can find collisions in the full version of sha1, requiring fewer than 2^69 operations. Likewise, MD5 hash streams are vulnerable by use of similar attacks. Furthermore, there are many online databases with millions of sha1 and MD5 records to look up the reverse stream. Available solutions have inherent vulnerability cases which are mentioned as follows.

X-509 certificate verification is vulnerable to resource exhaustion, included in X-509 certificate are public key used for digital signature verification. Choosing very large value for the public exponent and public modulus associated with an RSA public key causes the verification of that key to require large amounts of system resources. Therefore, a remote, unauthenticated attacker could consume large amounts of system resource on an affected device, thereby creating a denial of service. It is clear that server impersonation is possible wherever the messages for authentication steps are not sent encrypted.

The security of the OTP system is based on the non-invert ability of a secure hash function. Due to the infrastructure of LAMPORT OTP and use of MD5or SHA1 for encrypting the initial stream, the method is no longer secure as security flaws have been found in both hash function.

In addition, the result of OTP at each authentication time is based on a counter and the hash function; the method is no longer secure if a third party logs copies of previously entered values and result of hash.

HWANG OTP is also vulnerable. The cryptanalysis consists of three cases: eavesdropping, impersonation of server or client. In the eavesdropping base case the attacker records message between server and client in the ‘t’th and t+1th login stages. By using the recorded messages, the attacker then impersonates server. After receiving a response from client, the attacker can successfully impersonate client in the t+2th login stage. In step (1) the attacker records communication tokens in the t’th and t+1th login stages. In step (2) as impersonation of server case, by using the tokens recorded in step (1), the attacker computes the fake tokens for the t+2th login stage. With these tokens, the attacker plays the server's role for the t+2th login stage. The tokens are valid for the t+2th stage except that D_{t+1} is the previously used random number in the t+1th stage. Since the client dose not store the previous random number, he is unable to find out that the tokens are forged and thinks the attacker is a valid server. And finally at step (3) as impersonation of client case, the attacker tries to login to the server in the t+2th stage. After receiving the server's challenge, the attacker computes response. Since the attacker's response is exactly the same as the expected response to the server's challenge, the attacker is accepted as a valid client. Hence, the attacker succeeds in this so called re-play attack. To counter this scenario of attack, the client could store all the previous random numbers received from the server.

WPA2 or IEEE 802.11i amendment suffers from imperfect counter mode initialization which results in the collapse of whole security mechanism. The AES blocks are not transferred in a secure way anymore. The counter mode utilization can be regenerated. Therefore, not only the transferred information can be decrypted but also client and server impersonation as well.

As mentioned before, available solutions have vulnerabilities and they would not provide a promising secure environment for both sides (client/server). Totally, client and server of these technologies should concerned about their information and they are looking for better security service to store and transmit their important data in network's channel. Obtaining this significance is main reason for this study. A new approach was described with details in next chapter.

Chapter 3

MICROCONTROLLER-BASED IMPLEMENTATION OF PARSEKEY+

Producing a device as a pair of Key and Lock is known as an important concern in security science. In this research we wish to implement ParseKey+ on compatible AVR microcontroller supporting a USB connection to the host. Some AVRs provide built-in AES256 hardware module. ATmega32U4 [22] was chosen as the base microcontroller in this thesis. This family of microcontrollers has built-in USB 2.0 port. AES256, and other required functions could be also easily implemented.

This chapter presents the hardware implementation and application area of ParseKey+ algorithm. Finally, limitations and the method of overcoming these limitations will be studied.

3.1 ParseKey+

ParseKey+ is designed as an authentication mechanism of OTP nature with a twist. In successive logins, the strategy used in ParseKey+ does not change the key; but, the positions and lengths of scattered sub-keys in a uniformly distributed random-noise key file will change. Both the Client and the Server have their particular keys. Each is created from a seed by a one-way hash function at their respective sides. Verification of each side is done by extracting the sub-keys according to the positions (indices) and the lengths of sub-keys kept in the other side’s database.

Clearly this is in addition to the usual practice of username and password control. Furthermore, the password of each side is kept as hash hexadecimal stream, which is compared against the online hash generation of input stream. In this approach, a dual-side mechanism is used to meet secure authentication requirements.

The ParseKey+ Approach is an enhancement over the ParseKey [23]. The ordinary ParseKey approach is an authentication mechanism for client side only. ParseKey+ approach provides authentication mechanism for both client and server sides: the client is assured of server's genuineness, and, at the same time, the protocol guaranties that a legitimate client is communicating with the server. Thus, ParseKey+ guaranties trusted communication between a client and a server each being certain of the other's authenticity. As with OTP, ParseKey+ counters the most important shortcoming associated with the traditional passwords, namely the replay attack [5].In the following subsections, ParseKey+ will be detailed with respect to its key file structure, authentication approach, and formal specification.

3.1.1 The ParseKey+ Record Structure

Advanced Encryption Standard (AES), the successor to Data Encryption Standard was tested for possible vulnerabilities. The related-key rectangle attack with 4 related keys on 10-round reduced AES-256 requires a data complexity of 2^97.5 chosen plaintexts and a time complexity of about 2^254 encryptions [24] [25]; clearly, none of the presented attacks constitute a realistic threat to the security of AES. Therefore, we prefer to use AES instead of other alternative symmetric encryption methods – such as DES III– with same size of input and output stream for encryption of transferred messages over the net including ParseKey+ file.

In introducing the ParseKey+ authentication approach, its database requirements and key generation process are important. It should be noted that the term “key” is used to mean the string of hexadecimal characters output by a hash function. The generated key is also used to encrypt messages including ParseKey+ file by AES 256 algorithm for transfer over the net. This encryption algorithm requires 256-bit key length. Therefore, SHA-256 hash function was preferred for key generation as it produces the same bit sequence as required for AES 256. Client & Server, each side is aware of its password, and its hash is kept at the other side. Therefore, as mentioned earlier, hash of password is used as key for encryption of messages by AES. For this purpose, hash of password produced by SHA-256 provides 32 bytes (or 256 bits) key for bidirectional AES 256 bits cryptosystem [26].

In addition, the term “ParseKey+ file” stands for the file that hides the key (as sub-key segments distributed cyclically) among random noise sequences of hexadecimal characters produced by the same hash function used to produce the key itself. This practice assures uniform distribution of characters in the file indistinguishable whether they belong to the key or the filler. Use of same hash function counters ciphertext-only attack as the key and the random noise is of the same nature [27]. Uniformly distributed random numbers with long interval are used as input of hash function instead of timestamp that follows a known sequence and thus likely to infuse vulnerability. The key is chopped into sub-key pieces, which are then scattered around in the ParseKey+ file. The position indices of the sub-keys in ParseKey+ file in addition to consequent sub-key lengths of each part are kept at the client (/server) database or in another memory device at the client (/server) side together with their corresponding symmetric one-way encrypted passwords. As shown in Error! Reference source not found. Records kept in database at either side consist of a unique username used as primary key, hash of the opposite side's password (SHA-256 (pass(C))), the Key of own side, the own side's password for that individual record, the sequence of index and length of sub-keys supposed to be expected at next login from ParseKey+ file transferred by the opposite side, and finally encrypted ParseKey+ file of the opposite side [26].

Username can be presented as a sequence of digits. Structure of "IL" is as sequence of a pair of values indicating the position of a sub-key in the ParseKey+ file and the sub-key length. The ';' delimiter presents information belonging to individual sub-keys followed by position (Index) in ParseKey+ file, delimiter ',' and then length of sub-key. Textual values shown in table are for the sake of this example. For instance, the sequence "5,4;173,7…" indicates that the first sub-key starts at 5th byte in ParseKey+ file and has a length of 4 bytes, then second sub-key starts at byte 173 with length of 7 bytes. The following table represents a sample list of client login records at a server. Similar record structure is used at the client side.

Table 3.1: Login table: Sample client login records at server side

Username	SHA-256 (pass$_{(C)}$)	pass$_{(S)}$	Key$_{(S)}$	IL$_{(S)}$	Encrypted ParseKey$_{(C)}$ (EP$_{(C)}$)
UYSAL	f17df15e…	1256953732	0760c…	5,4;173,7;260,5;475,13;599,9	913b04c54574d18…
RAHNAMA	e7732b9…	1257552359	46e6a…	27,2;312,9;729,15;900,14	cab7aec4a8c590a1…
MAKVANDI	94b321e…	1296920027	ecd81…	527,12;690,2;739,18;37,8	c8e14d7106e83bb…
……	……	……	……	……	……

Notice that in creating the fixed-size ParseKey+ file, sub-keys are randomly distributed in the file in any order. For example, for the user UYSAL (Table 1) it is made as follows: the first 4 symbols of Key(S) should match 4 hexadecimal characters in ParseKey(S) (available at client side database, which is sent to server side for matching) starting at the position number 5 (“0760” should be there, next 7 symbols in Key(S) should match the 7 symbols in ParseKey(S) starting at the position number 173, etc.

3.1.2 The ParseKey+ Authentication Approach

The ParseKey+ authentication approach involves both user and server actively. At the beginning of the communication for authentication, client interacts with the server, supplying the client’s username concatenated with timestamp, and the password altogether encrypted by server name. The server authentication service verifies the client-provided data against that kept in the server DB; if matched, the server sends back the server-name with the new timestamp and a uniquely generated server-side password for that client, which is encrypted by username. If the client can verify the server’s data then it returns an acknowledgement message. Having received acknowledgement, the server returns the client’s decrypted ParseKey+ file (which was kept in DB at last login after having encrypted using the client’s password hash). The client uses the ParseKey+ file to retrieve the key and compare it against the key kept in client’s memory device [28].

If they match, then the client returns server’s decrypted ParseKey+ file (which was similarly encrypted using server’s password hash). If the key extracted from this file by the server matches the one saved in server’s Database, then a new ParseKey+ file is encrypted using hash of the client’s password, and then downloaded to the client for use in future logon. The client reciprocates this by uploading the server’s new

ParseKey+ file encrypted using hash of server's password for use by the server in future logon of this client. Finally, server allows client entry into the system. This proves to either party that the other side is authenticated [26].

3.2 Possible Attacks

Here we discuss attacks and vulnerabilities of ParseKey+ authentication approach. Among the well-known attacks Replay attack, Meet-in-the-middle (MITM), Ciphertext-only, and Side channel attack seem to be more of interest based on structure of ParseKey+.

3.2.1 Replay Attack

Attacker replay attack by an unsecured computer into achieve access where an authentication period .As mentioned earlier, ParseKey+ is of OTP nature without reusing previously made key in generating new key. The strategy used in ParseKey+ follows the idea behind OTP where the key for next login is changed. Therefore, it counters the replay attack.

3.2.2 MITM

Meet-in-the-middle attack is used against cryptographic algorithms with multiple keys for encryption. ParseKey+ messages are encrypted by AES 256 bits and then transferred to the other side. The encryption key is hash of other side's password. The other side applies hash to its known password and decrypts the message. Therefore, only one key is used for both encryption and decryption and in such scheme MITM is not applicable.

3.2.3 Ciphertext-Only

Sub-key segments are scattered in ParseKey+ file among random noise sequences of hexadecimal characters produced by the same hash function used to produce the key itself. This practice assures uniform distribution of characters in the file whether they belong to the key or the filler. Use of same hash function counters ciphertext-only attack as the key and the random noise is of the same nature.

3.2.4 Side-Channel Attack

A security hole might be countered where timestamp is used to produce random noise by hash function in creation of ParseKey+ file. Use of time stamp or any data that follows a known sequence causes vulnerability of side-channel attacks. Utilizing a uniformly random number generator with long interval will immunize the scheme against side-channel attack.

3.3 Hardware Implementation

The Atmel AVR family includes huge number of microcontrollers with varying features and packaging. The implementation of a hardware platform supporting various security protocols and functions needed in ParseKey+ in addition to providing a communication media to interchange data with PC requires a powerful yet inexpensive controller.

Firstly, ATmega32 [29] was chosen as main microcontroller. This microcontroller is one of the most famous ones in the family of 8 bit Atmel AVR products and it is compatible with many protocols we require while implementing ParseKey+ algorithm. Therefore, the board was designed to support SD card and USB connection in addition to some other peripherals for a general purpose development board. Schematic and PCB are illustrated in figure 3.1 and 3.2. SD card is used to keep ParseKey+ table data as shown in table 3.1.

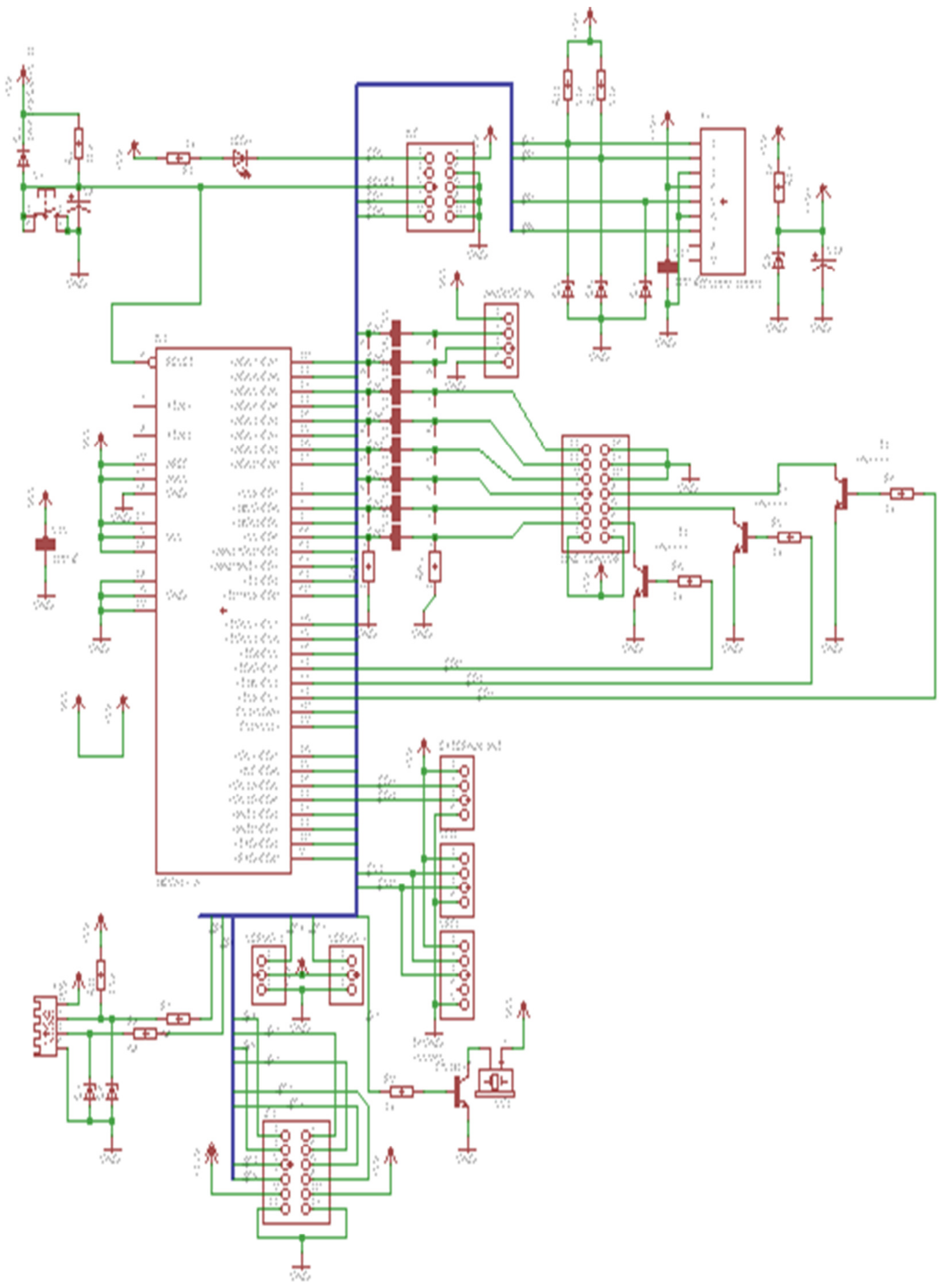


Figure3.1: schematic of development board.

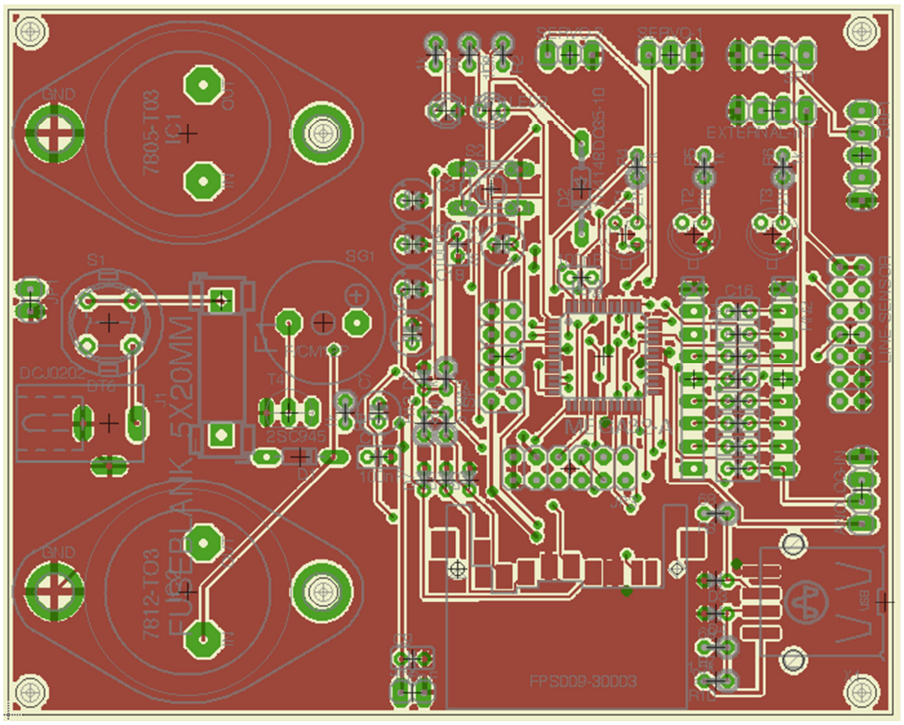


Figure 3.2: schematic of PCB development board.

However, the board which was implemented had problem in size which was not very suitable for pocket-size security keys. In addition, The USB protocol on ATmega32 microcontroller was not fully compatible with the host OS.

In this respect, some other microcontrollers from AVR 8bit family were elected. ATmega32U4 is the best candidate among all choices. This inexpensive microcontroller supports USB protocol naturally [30]. The table of parametric comparison among all microcontrollers that was chosen for this work is given in table3.2.

Table 3.2: Microcontroller parametric table [30].

Device	Flash (Kbytes)	EEPROM (Bytes)	SRAM (Bytes)	F.max (MHz)	ISP	Self-Program	SPI	UART
AT90USB162	16	512	512	--	--	--	--	--
ATmega162	16	512	1024	16	Yes	Yes	1	2
ATmega16U2	16	512	512	16	Yes	Yes	1	--
ATmega16U4	16	512	1536	16	Yes	Yes	Yes	Yes
ATmega328	32	1024	2048	20	Yes	Yes	1+USART	1
ATmega32U2	32	1024	1024	16	Yes	Yes	1	--
ATmega32U4	16	1024	3072	16	Yes	Yes	Yes	Yes
ATmega8A	8	512	1024	16	Yes	Yes	1	1
ATmega8U2	8	512	512	16	Yes	Yes	1	--

The ATmega32U4 is a low-power CMOS 8-bit microcontroller based on the AVR enhanced RISC architecture. By executing powerful instructions in a single clock cycle, the ATmega32U4 achieves throughputs approaching 1 MIPS per MHz allowing the system designed to optimize power consumption versus processing speed. Figure 3.1 shown schematic pin out ATmega32U4 micro controller. Simplified block diagram for the ATmega32U4 shown in figure 3.2 [22].

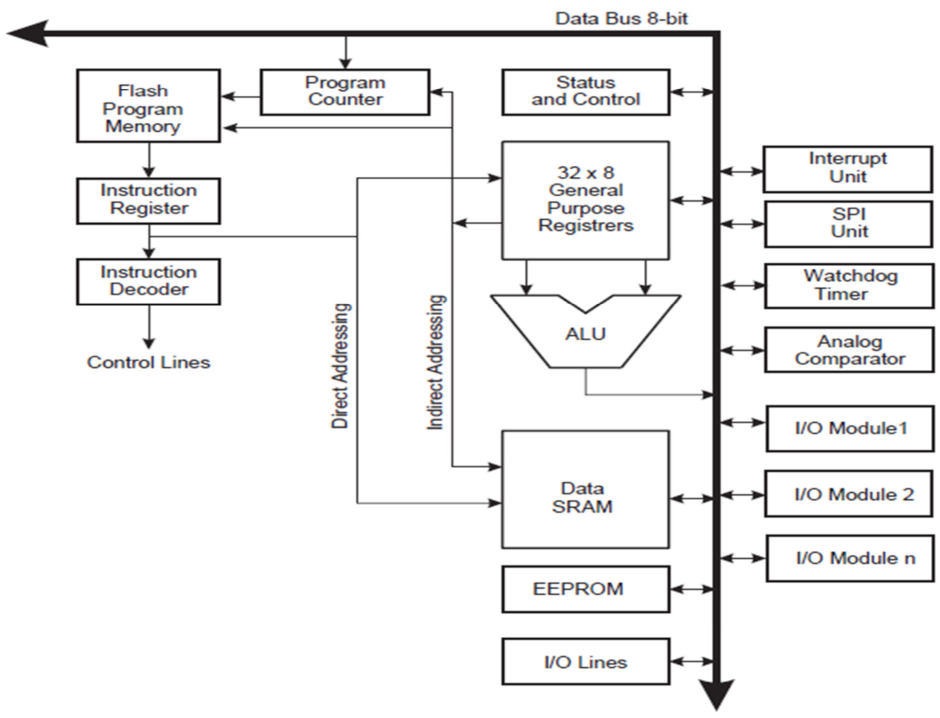


Figure 3.3: Block Diagram Architecture of the ATmega32U4 **[22]**.

3.4 Limitations

An 8 bit RISC microcontroller working on 8~20 MHz is not a very powerful and suitable device for calculation of AES or SHA security algorithms. Therefore, implementation of such algorithms requires specific programming techniques for memory and other resources management. For instance, ATmega32U4 contains about 2.5KB of SRAM which is very limited for keeping even ParseKey+ record structure. On the other hand, each security certificate (each server) generates a separate and unique record on the client side. Therefore, USB dongles need a tiny database system to save and retrieve such records on an embedded flash memory.

On the other hand, such USB dongle should be able to communicate with the host computer, keeping the ParseKey+ record structure on either of local or embedded databases, and processing different security algorithms.

Among those candidates, the fastest processor with maximum amount of SRAM memory and those ones supporting USB are preferred. In addition, the smallest footprint database available for embedded systems should be used. Secondly, such optimized database management systems for 8 bit AVRs are not found in the market yet. The smallest available DB (SQLite) requires about 100KB of RAM at least. Thirdly, the data record files accessed by database should be kept on FAT16/32 based flash memory securely so that accessing the flash memory offline does not entail the contents of ParseKey+ file.

As result, the 32 bits File Allocation Table (FAT32) with secure master boot record and partition table is used to keep ParseKey+ data file. Currently MMC and SD cards support internal encryption which can be activated to enhance to overall security against offline attacks. In the next chapter nominated design for hardware and codes for database and file system in addition to implementation of base functions is given.

Chapter 4

IMPLEMENTATION

This chapter focuses on the microcontroller based implementation of ParseKey+ and its related necessary modules including a compact database, secure and compatible file system, SHA256 and AES256 modules, and finally ParseKey+ protocol.

4.1 Implementation Detail

Microcontroller based implementation of ParseKey+ contains two main phases, namely, software and hardware. Following we provide detailed explanation on required parts of both phases.

4.1.1 Data Store and Retrieval

As mentioned earlier, ParseKey+ data file contains ILs, keys, etc. In this process a record of information needs to be retrieved and updated to the new shape. The record structure can be kept as raw data on disk in a file. Variable Field (VF) and Delimited Field (DF) approaches are both suitable for implementation of the record structure. In both cases it is required to keep the file on an embedded disk and make it accessible for future updates. Therefore, file and directory structure, create and remove file functions, and read and write functions are implemented with limited buffer size while storing or retrieving raw data on disk due to limited SRAM memory capacity.

On the other hand, micro SD Card is used to keep the ParseKey+ data file. ParseKey record structure is limited to few kilobytes and therefore 1GB SD card may contain few hundred thousands of unique credentials without any problem. FAT32 is a widely used file system. Several code implementations were tested on Atmega32U4 microcontroller and finally an AVR Studio compatible code was chosen. The reason to choose AVR Studio as the compiler were firstly the standard main stream Atmel company has provided, secondly, many developers provide their codes in this platform, finally it is an open-source compiler, free of charge providing an on-chip debugger functionality.

Therefore, universal code source should be chosen to let us modify codes for using in this research (Dharmani's code source was selected and modified). Main library of modified codes were shown in the appendix A.

4.1.2 Database

Today over 98% of all available microprocessors are part of embedded systems [31], [32]. ParseKey+ scheme needs a database system to select/update or delete ParseKey+ records. Quite a few ready DBMS software are available which can manage the server side requirement. However, the smallest footprint database for embedded applications requires about 100KB or SRAM for BTree indices and inter-buffer exchanges.

To overcome this problem, we tested various databases specifically designed for embedded systems. It is noticed that there is no single available DBMS for small footprint microcontrollers such as the one we use. Smart Card DBMS [33], Stones DB [34], IBM DB2 Everyplace [35], LGeDBMS [36], and TinyDB [37] were checked for compatibility. Each system has been developed to serve a specific application. Moreover, each of above-mentioned database systems is designed for a specific embedded platform and it does not work on any other hardware. Therefore, designing a universal and efficient database supporting 8 bit AVR microcontrollers is necessary.

4.1.3 Teeny2 USB Development Board

Among various candidates for small size however powerful and inexpensive hardware against the design we had provided earlier, Teensy2 is considered as the best choice.

The Teensy2 USB Development Board is a complete USB-based microcontroller development system. In this board ATmega32U4 used as microcontroller. This microcontroller embeds an internal serial to USB convertor [38]. Figure 4.1 shows Teensy2 development board.

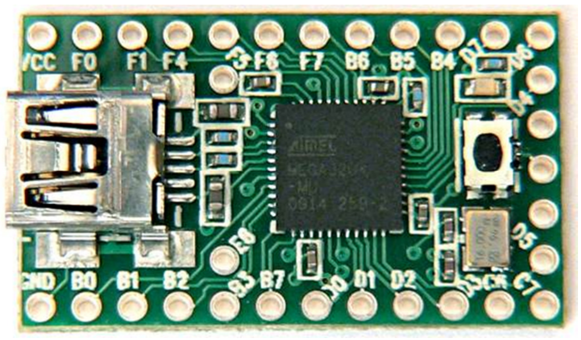


Figure 4.1: Teensy 2 development board.

Teensy board has separate Micro SD adaptor. This adaptor allows designer to use Gigabytes of flash memory with development board. Figure 4.2 shows separate SD adaptor and installed board shown in Figure 4.2.

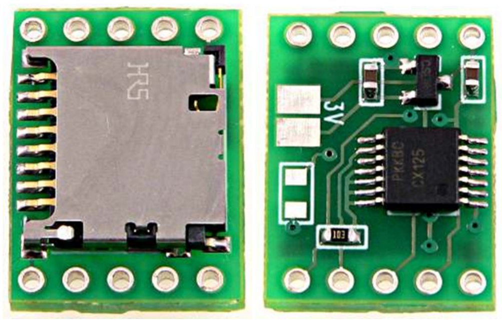


Figure 4.2: SD Adaptor.

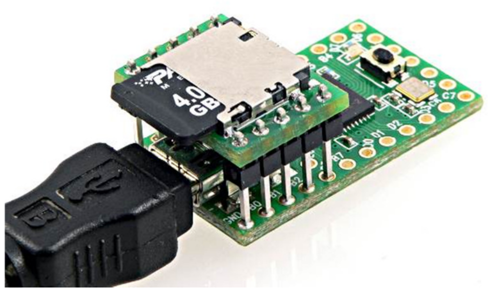


Figure 4.3: Teensy board with SD Adaptor.

4.1.4 AVR Boot Loader

It is necessary to update the firmware several times during development phase. Teensy2 allows on-board programming of the chipset without necessarily unplugging it. Teensy2 development board has special bootloader (the modified version of the bootloader provided by Atmel) to flash or rewrite the program on the microcontroller. Teensy2 bootloader only requires synchronizing with compiler to execute the compiled code on ATmega32U4 microcontroller.

4.1.5 Program

Underlying operations of the ParseKey+ approach may be precisely formulated in a new representation scheme that will be introduced in Table 4.1

Table 4.1: Legend for formulation of ParseKey+ approach **[28]**.

Legend	**Meaning**
.	String concatenation
(C) as subscript	Pointer to client's indices
(S) as subscript	Pointer to server's indices
{M}	Message M
//	Comment till end of line
$\rightarrow${M}	Transfer of control flow and message (M)
Acknowledge	A message to say that the sent data was received intact
Begin	Begin of body of function
C, C:	Client, Client side
Check Availability (name)	Checks if the username or server name is available to register
Check Validity (info)	Checks if the information matches with that kept in DB
Concatenate (vector, array)	Inserting the vector after the last row of the array
CreatePK ()	Return a newly created and saved ParseKey+ file
Decrypt (C,K)	Symmetric decryption of the ciphertext C using AES 256/… with the key K
Encrypt (P,K)	Symmetric encryption of the plaintext P using AES 256/… with the key K
End	End of body of function
EP(x)	Encrypted ParseKey+ file kept in DB or memory
FetchArrayElement (IL(x))	Returns the first element of vectors of [Index, Length] from IL and removes it from the queue. Returns 0 if no item remaining
Hash ()	A hash function such as SHA-256
IL	Array of vectors [Index, Length]
MakePK_File (key, IL)	Scattering sub-keys among random noise in ParseKey+ file based on values of IL
OTP ()	Generates one time password for use next login
PK_File_size	ParseKey+ File defined size: Key length plus random noise. The random noise can

	be varied from 0 byte to (PK_File_size – key size) bytes.
Random (x)	Generating an integer random number between 0 and the input value x
ReturnKey (PK, IL)	Merging all sub-keys retrieved from ParseKey+ file using index and length pairs to obtain and return the key.
S, S:	Server, Server side
Save (info)	Inserting or updating the information as a record in DB or a Memory device
Sizeof (stream)	Size of a stream in terms of bytes
Terminate	Rejects and exits the process

Using the nomenclature of Table 4.1, processes of ParseKey+ approach are defined below in terms of the first time and subsequent logons. Firstly definitions of some prominent utility functions are given as follows:

Creation ParseKey+ function:

```
 Function CreatePK (x)
 // creates ParseKey+ file and stores its corresponding IL in
 DB
 Begin
 RS(x) = Random (Timestamp);
 Key(x) = sha256 (RS(x));
 PrevIndex=0;
 PrevSubKeySize=0;
 while (Sizeof (Key(x) - PrevSubKeySize)>0)
 Begin
 Index = Random (PK_File_size - PrevIndex +
 PrevSubKeySize);
 PrevIndex=Index;
 Length = Random (Sizeof (Key(x)-PrevSubKeySize);
 PrevSubKeySize= PrevSubKeySize + Length;
 IL(x) = IL(x) Concatenate (Vector [Index, Length]);
 End
 ParseKey(x) = MakePK_File (Key(x), IL(x));
 Save (Key(x), IL(x));
End
```

```
Function MakePK_File (Key(x), IL(x))
//Scattering sub-keys among random noise in ParseKey+ file based on
values of IL

 Begin
 ParseKey(x) = 0;
 HashLength = Sizeof (sha256 (Timestamp));
 Iteration = PK_File_size / HashLength;
 for i = 0 TO i < Iteration
 Begin
 ParseKey(x) = ParseKey(x). sha256 (Random seed);
 End
 while((Current_IL=FetchArrayElement (IL(x)))! =0)
 Begin
 Overwrite (ParseKey(x), Current_IL, Key(x));
 End
 End
```

```
Protocol OTP ()
// one time password based protocol for exchanging ParseKey+ file to be
used at next login

 Begin
 C: ParseKey(C) = CreatePK (C);
 C: m = AESEncrypt (ParseKey(C), SHA256 (password(C)));
 C_S {m};
 S: Save EP(C);
 S: ParseKey(S) = CreatePK (S);
 S: m = AESEncrypt (ParseKey(S), SHA256 (password(S)))
 S_C {m};
 S: Set session values to let client access resources
 C: Save EP(S);
 End
```

First Time Login (registration):

```
C: m = AESEncrypt ({username. timestamp, SHA256 (password(C))}, server-
name);
C_ S {m};
S: AESDecrypt ({m}, server-name);
S: If not CheckAvailability (username)
 Terminate;
 Else
 Save (username, SHA256 (password(C)) ;
S: m = AESEncrypt ({server-name. timestamp, SHA256 (password(S))},
username);
S_ C {m}; // password(S) is unique for each client
C: AESDecrypt ({m}, username);
C: If not CheckAvailability (server-name) or timestamp is same
 Terminate;
Else
 Save (server-name, SHA (password(S)) ;
C: OTP ();
```

Subsequent Logins:

```
C: m = AESEncrypt ({username. timestamp, SHA256 (password(C))}, server-
name);
C_ S {m};
S: AESDecrypt ({m}, server-name);
S: If CheckValidity (username, SHA256 (password(C)))
S: m = AESEncrypt ({server-name. timestamp, Hash (password(S))},
username);
S_ C {m};
Else
 Terminate;
S: AESDecrypt ({m}, username);
C: if CheckValidity (server-name, SHA256 (password(S)))
C_ S {Acknowledge};
Else
 Terminate;
S: m = AESDecrypt (EP(C), SHA256 (password(C)));
S_C: {m};
C: KeyTemp = ReturnKey (ParseKey(C), IL(C));
If not (CheckValidity(KeyTemp))
 Terminate;
C: m = AESDecrypt (EP(S), SHA256 (password(S)));
C_S {m};
S: KeyTemp = ReturnKey (ParseKey(S), IL(S));
If not (CheckValidity (KeyTemp))
 Terminate;
```

```
S_C {Acknowledge};
C: OTP ();
```

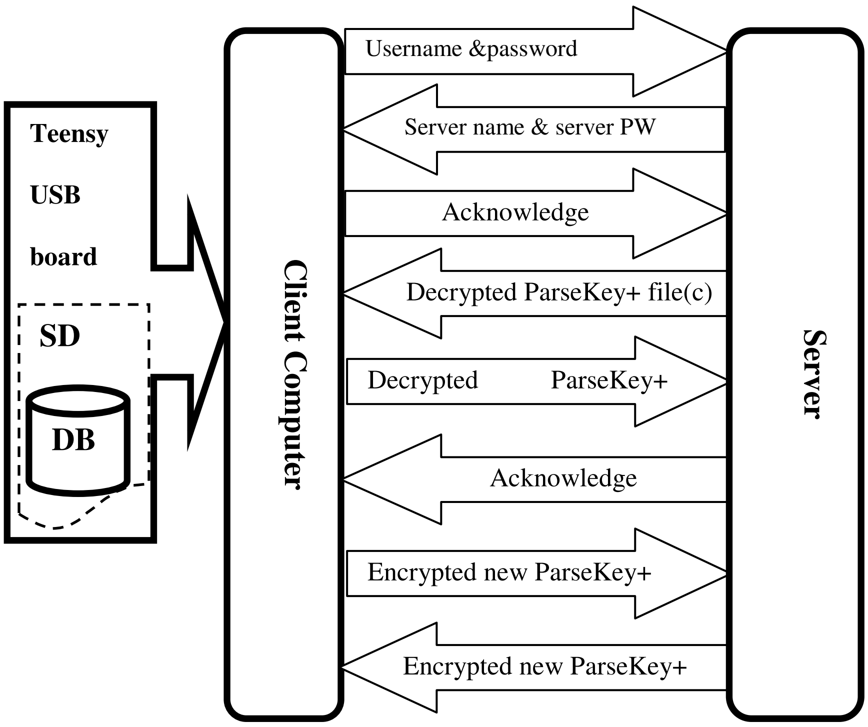


Figure 4.4: A simplified overview of exchanges during subsequence login.

In figure 4.4 Microcontroller based dongle running ParseKey+ based authentication during subsequent login was illustrated. See Appendix A.

As mentioned in ParseKey+'s algorithm, hashing function and encryption service are main parts in main body. In stage of first login SHA256 function was used as hashing function to hash password in both side. SHA256 was introduced in chapter 2 has many standard source codes. Although, a code was needed what it can communicate with main application. Therefore, source code it should be modified with Parsekey+ application. This story was repeated for encryption protocol. Firstly, for encryption and decryption of transmitted message through the channel between client and server, AES256 was selected. This protocol likewise required modifying to synchronize with ParseKey+ application. Codes of these functions were illustrated in appendix A.

4.1.6 Compiler

Application source codes were provided in C language. They should be compiled all together as a single project to be run on microcontroller. Atmel has indicated many solutions for AVR microcontrollers including CodeVision CAVR Compiler, FastAVR, IAR Systems and AVR studio. These compilers support programming for most of available Atmel AVR chips. However due to the nature of open-source programming, available implementations of other required software modules are found in different compilers. Therefore, it is necessary to unify the code so that all parts of the implementation can be merged into a single project compiled using a single compiler. AVR studio was selected as the compiler, due to aforementioned requirements. In addition, Teensy2 is designed very compatible with AVR Studio

4.2 Comparison

Many approaches in authentication services require a hash function to transpose sub-keys. Moreover, the password itself is kept as a hash stream in a database. An analysis to compromise the full SHA1 was reported in February 2005 [39]. Likewise, MD5 hash streams are vulnerable to similar attacks. Furthermore, there are many online databases with millions of SHA1 and MD5 records to lookup the reverse stream [40].

According to sections 2.9 and 2.10 about available solutions and their vulnerabilities and possible attacks were described in section 3.3, summarize comparison illustrated in table 4.2

Table 4.2: Comparative cipher analysis of other schemes and ParseKey+ [23].

	X.509	**Lamport OTP**	**Yeh-Shen-Hwang OTP**	**Public-Key Cryptosystem**	**ParseKey+**
Complexity	Very high	High	High	Low	Low
Decryption of payload	possible	Not possible	Not possible	Not possible	Not possible
Client impersonation	possible	possible	possible	possible	Not possible
Server impersonation	possible	possible	possible	possible	Not possible
Replay attack	Not possible	possible	Not possible	Not possible	Not possible
MITM	possible	Not possible	Not possible	possible	Not possible
Ciphertext-only	possible	possible	possible	possible	Not possible
Side channel attack	possible	possible	possible	Not possible	Not possible

As in Table 4.2 is determined, ParseKey+ approach with low complexity can better Resistance against different type of attacks. Decryption of transferred data is impossible. Furthermore, it prevents client and server impersonations in addition to avoiding replay attack while the OTP systems are shown to be vulnerable. The space-usage Reasonable cost efficiency of ParseKey+ is rather low for data transfer purpose in comparison.

In RFID-based implementation ParseKey+ [26] likewise has limitation in memory to allocate for provided data file in ParseKey+ process. Furthermore, processing speed in RFID-based implementation ParseKey+ is very low (time for each process is approximate 10 second). These limitations overcome in microcontroller-based implementation. In this study limitation of memory was overcame with use extra SD-card and in processing time for microcontroller like ATmega32U4 is highly than RFID reader.

Chapter 5

CONCLUSION AND FUTURE WORK

In order to increase the security of an authentication service, procedures in addition to exchanging username & password are needed. This study tries to implement ParseKey+, a multi-way strong authentication procedure, as an operative approach to mutual client / server impersonation avoidance into microcontroller. ParseKey+ is improved by using transposition of scattered sub-keys in uniform random noise spread over a ParseKey+ file. The files for user and server are changed at each user session; therefore, it also guarantees unique login in addition to countering replay attack.

The new approach has less complexity than other OTP schemes, and decryption of transferred information is infeasible. More importantly, it avoids client and server impersonations in addition to avoiding replay attack while the OTP schemes are shown to be vulnerable.

Compared to other approaches, the space-usage wise cost efficiency of ParseKey+ is rather low for data transfer purpose. This is due to redundancy introduced into the ParseKey+ File. However, the superb security provided by ParseKey+ is desired in certain operations requiring heightened concerns such as key exchange and is cases where fake server may not be tolerable.

Comparative cipher analysis indicates that ParseKey+ performs better than current OTP schemes. Additionally, we believe that ParseKey+ provides better security in less computation time in comparison against X.509 based authentication schemes.

This development board can be used as a security dongle for servicing better and safer authentication instead of traditional services and smart cards. In addition, clients can merge all their smart cards and keys into one unit. This opportunity decreases associated separate documents and electronic gadgets, thereupon increase security.

As mentioned before, this implementation designed to connect with plug-in service or a host program to the operating system. We wish to write similar applications for other operating systems in future.

As the future work, we wish to enhance ParseKey+ scheme using steganography method. We would like to merge new approach and steganography method and hide data into a multimedia package (streamed video, photo, voice etc.) rather than keeping the information in ParseKey+ data file.

REFERENCES

[1] President's Information Technology Advisory Committee. Cyber Security. Retrived on September 2010 http://www.itrd.gov/pitac/reports/20050301_cybersecurity/cybersecurity.pdf.

[2] Rahnama.B, Elci.A, "Considerations on a new Software Architecture for Distributed Environments using Autonomous Semantic Agents," in *Compsac 29*, Edinburgh, UK, 2005, pp. 133-138.

[3] Elci.A, Rahnama.B and Amintabar.A, "Tracking Reported Vehicles in Traffic Management and Information System using Intelligent Junctions," *Vehicular Wireless Networks and Vehicular Intelligent Transportation Systems, Journal of Information Science and Engineering*, pp. 883-896, 2010.

[4] Litwin.E, "Cryptography," *International Journal of Potentials*, pp. 36-38, 2001.

[5] Atkinson.N, Haller.R, RFC 1704:On Internet Authentication. Retrived on June 2010. http://www.ietf.org/rfc/rfc1704.txt

[6] Stallings.W, *Network security essentials.*: Prentice Hall, 2003.

[7] Prepared by the office of the manager, *public switched network security assessment guidelines.*: Diane publishing, 2002.

[8] Nicholas.M, Orlans.P, Higgins.T ,Woodward.J, *Biometric and strong authentication.*: McGraw-Hill Inc, 2003.

[9] Hallam-Baker.J, Hostetler.P, Lawrence.J, Leach.S, Luotonen.P, Stewart.A, Franks.L, "HTTP Authentication: Basic and Digest Access Authentication," *ITEF*, 1999.

[10] Metz.N, Nesser.C, Straw.P, Haller.M, "A One-Time Password System," *IETF*, pp. 2-4, 1998.

[11] Vazquez.M, An Overview and Competitive Analysis of the One-Time Password (OTP) Market. Retrived on October 2010. https://rsaemail.rsa.com/servlet/campaignresponden

[12] Stallings.W, *Cryptography & Network Security: Principles & Practice*, 2nd ed. USA: Prentice Hall, 1998.

[13] Schneider.B, *Applied Cryptography: Protocols, Algorithms, and Source Code in C* , 2nd ed.: John Wiley and sons.

[14] Ashley.P, *Practical Intranet Security: Overview of the State of the Art and Available Technologies*.: Kliwer Academic publishing.

[15] Seleborg.S, About AES – Advanced Encryption Standard. Retrived on November 2010. http://www.axantum.com/axcrypt/etc/About-AES.pdf

[16] Lotfy.O, Noureddi.S, Ahmet.M, Camel.B, Mourad.T, "AES Embedded Hardware Implementation," in *Second NASA/ESA Conference on Adaptive Hardware and Systems*, 2007.

[17] McCurley.K, "A Fast Portable Implementation of the Secure Hash Algorithm," , 2001.

[18] Wang.L, Aoki.K, Sasaki.Y, "Preimage Attacks on 41-Step SHA-256 and 46-Step SHA-512," in *NTT Information Sharing Platform Laboratories, NTT Corporation*, 2008.

[19] Johnson.F, Steganography Technical Report. Retrived on September 2010. http://www.jjtc.com/stegdoc/sec202.html

[20] kessler.G, "Steganography: Hiding Data Within Data," *Windows &.NET* , 2002.

[21] Rabah.K, *Using samba 3 client technology and kerberos for window active directory-based identity management.*: GTS institute, Retrived on September 2010. http://www.docstoc.com/docs/53461936/

[22] Atmel products. Retrived on November 2010. http://www.atmel.com/dyn/resources/prod_documents/7766S.pdf

[23] Elçi.A, Rahnama.B, "ParseKey+: a Five-Way Strong Authentication Procedure as an Approach to Client/Server Impersonation Avoidance Using Steganography for Key Encryption," in *International Conference on Security and Management (SAM'07)*, 2007, pp. 25-28.

[24] Kim.J, Hong.S, and Preneel.B, "Related-key rectangle attacks on reduced AES-192 and AES-256," in *FSE2007,LNCS4593*, 2007, pp. 225-241.

[25] Wei.Y, Hu.Y, "New related-key rectangle attacks on reduced aes-192 and aes-256," in *Science in China SeriesF: Information Sciences*, 2009, pp. 617-626.

[26] Rahnama.B, Elci.A, Celik.S, "Securing RFID-Based Authentication Systems Using ParseKey+," in *3rd International Conference on Security of Information and*

Networks (SIN'10), Taganrog, Rostov-on-Don, Russian Federation, 2010, pp. 212-217.

[27] Biryukov.A ,Kushilevitz, "From Differential Cryptanalysis to Ciphertext-Only Attacks," in *CRYPTO*, 1998, pp. 72-88.

[28] Rahnama.B, Elçi.A, "AWGN based Seed for Random Noise Generator in ParseKey+," in *Internatonal Conference on Security of Information and Networks (SIN 2009)*, 2009, pp. 244-248.

[29] ATmega32u4 datasheet. Retrived on December 2010. http://www.atmel.com/products/avr/default.asp?family_id=607&source=global_nav

[30] Atmel Comparison table. Retrived on December 2010. http://www.atmel.com/dyn/products/product_card.asp?part_id=4317

[31] Krishnan.R, "Future of Embedded Systems Technology," in *Technical Report GB-IFT016B, BCC Research*, 2005.

[32] Tennenhouse.D, "Proactive Computing," in *Communications of the ACM*, 2000, pp. 43-50.

[33] Bouganim.L, Pucheral.P, Anciaux.N, "Smart Card DBMS: where are we now? Technical Report 80840," 2006.

[34] Ganesan.D, Mathur.G, Shenoy.P,Diao.Y, "Rethinking Data Management for Storage centric Sensor Networks," in *In Proceedings of the Conference on Innovative Data Systems Research (CIDR)*, 2007, pp. 22-31.

[35] Lal.A, Leung.C, Pham.T, Karlsson.J, "IBM DB2 Everyplace: A Small Footprint Relational Database System," in *In Proceedings of the International Conference on Data Engineering (ICDE)*, 2001, pp. 230-232.

[36] Baek.S, Lee.H, Joe.M, Kim.J, "LGeDBMS: a Small DBMS for Embedded System with Flash Memory.," in *In Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 2006, pp. 1255-1258.

[37] Franklin.M, Hellerstein.J, Hong.W, Madden.S, "TinyDB: An Acquisitional Query Processing System for Sensor Networks," *ACM Transactions on Database Systems (TODS*, pp. 122-173, 2005.

[38] Teensy2 solution. Retrived on October 2010. http://www.pjrc.com/store/teensy.html

[39] Yin.x, Yu.y, Wang.h, "Finding Collisions in the Full SHA-1," in *CRYPTO '05, Lecture Notes in Computer Science*, 2005, pp. 17-36.

[40] Ramsey.b, Reverse MD5 Hash Lookup. Retrived on September 2010. http://md5.benramsey.com/

[41] FIPS PUB 198-1, "The Keyed-Hash Message Authentication Code (HMAC)," *National Institute of Standards and Technology*, 2007.

[42] Karl, Ware, *biometric and strong authentication.*: McGraw-Hill Osborne, 2002.

[43] TU.Y, Gao.Z, "An improvement of dynamic ID based remote user authentication scheme with smart cards," in *7th World Congress on Intelligent Control and Authentiation*, 2008.

[44] NIST. fips-197: Advanced Encryption Standard. Retrived on December 2010. http://csrc.nist.gov/encryption/aes/

[45] Haller.N, The S/KEY One-Time Password System. Retrived on September 2010. http://www.ietf.org/rfc/rfc1760.txt

APPENDICES

APPENDIX A

AES header library:

```
/* aes.h */
* \file aes.h
 * \email daniel.otte@rub.de
 * \author Daniel Otte
 * \date 2008-12-30
 * \license GPLv3 or later
#ifndef AES_H_
#define AES_H_
#endif
```

SHA256 header library

```
/* sha256.h */
* \file sha256.h
 * \author Daniel Otte
 * \date 2006-05-16
 * \licens GPLv3 or later
#ifndef SHA256_H_
#define SHA256_H_
#define __LITTLE_ENDIAN__
#include <stdint.h>
/** \def SHA256_HASH_BITS
 * defines the size of a SHA-256 hash value in bits
 /** \def SHA256_HASH_BYTES

 * defines the size of a SHA-256 hash value in bytes

 /** \def SHA256_BLOCK_BITS

 * defines the size of a SHA-256 input block in bits

 /** \def SHA256_BLOCK_BYTES

 * defines the size of a SHA-256 input block in bytes

#define SHA256_HASH_BITS 256

#define SHA256_HASH_BYTES (SHA256_HASH_BITS/8)

#define SHA256_BLOCK_BITS 512

#define SHA256_BLOCK_BYTES (SHA256_BLOCK_BITS/8)

/** \typedef sha256_ctx_t

 * \brief SHA-256 context type

* A variable of this type may hold the state of a SHA-256 hashing process typedef struct {

 uint32_t h[8];

 uint64_t length;

} sha256_ctx_t;

/** \typedef sha256_hash_t

 * \brief SHA-256 hash value type

* A variable of this type may hold the hash value produced by the

 * sha256_ctx2hash(sha256_hash_t* dest, const sha256_ctx_t* state) function. typedef uint8_t
sha256_hash_t[SHA256_HASH_BYTES];

/** \fn void sha256_init(sha256_ctx_t *state)

 * \brief initialise a SHA-256 context

* This function sets a ::sha256_ctx_t to the initial values for hashing.

 * \param state pointer to the SHA-256 hashing context

void sha256_init(sha256_ctx_t *state);

/** \fn void sha256_nextBlock (sha256_ctx_t* state, const void* block)

 * \brief update the context with a given block

 * This function updates the SHA-256 hash context by processing the given block of fixed length.

 * \param state pointer to the SHA-256 hash context

 * \param block pointer to the block of fixed length (512 bit = 64 byte)
 void sha256_nextBlock (sha256_ctx_t* state, const void* block);
/** \fn void sha256_lastBlock(sha256_ctx_t* state, const void* block, uint16_t length_b)
 * \brief finalize the context with the given block
* This function finalizes the SHA-256 hash context by processing the given block
 * of variable length.
 * \param state pointer to the SHA-256 hash context
 * \param block pointer to the block of fixed length (512 bit = 64 byte)
 * \param length_b the length of the block in bits
void sha256_lastBlock(sha256_ctx_t* state, const void* block, uint16_t length_b);
/** \fn void sha256_ctx2hash(sha256_hash_t* dest, const sha256_ctx_t* state)
 * \brief convert the hash state into the hash value
 * This function reads the context and writes the hash value to the destination
 * \param dest pointer to the location where the hash value should be written
 * \param state pointer to the SHA-256 hash context
void sha256_ctx2hash(sha256_hash_t* dest, const sha256_ctx_t* state);
/** \fn void sha256(sha256_hash_t* dest, const void* msg, uint32_t length_b)
 * \brief simple SHA-256 hashing function for direct hashing
* This function automaticaly hashes a given message of arbitary length with
 * the SHA-256 hashing algorithm.
 * \param dest pointer to the location where the hash value is going to be written to
 * \param msg pointer to the message thats going to be hashed
 * \param length_b length of the message in bits
 void sha256(sha256_hash_t* dest, const void* msg, uint32_t length_b);
#endif /*SHA256_H_*/
```

FAT32 header library:

```
//*******************************************************
// **** ROUTINES FOR FAT32 IMPLEMATATION OF SD CARD *****
//*******************************************************
//Controller: ATmega32U4 (Clock: 16 Mhz-internal)
```

```
//Compiler: AVR-GCC (winAVR with AVRStudio)

//Version             : 2.4

//Author : CC Dharmani, Chennai (India)

// www.dharmanitech.com

//Date: 17 May 2010

//********************************************************

//Link to the Post: http://www.dharmanitech.com/2009/01/sd-card-interfacing-with-atmega8-
fat32.html

#ifndef _FAT32_H_

#define _FAT32_H_

//Structure to access Master Boot Record for getting info about partioions

struct MBRinfo_Structure{

unsigned char       nothing[446]; //ignore, placed here to fill the gap in the structure

unsigned char       partitionData[64];             //partition records (16x4)

unsigned int        signature;                     //0xaa55

};

//Structure to access info of the first partioion of the disk

struct partitionInfo_Structure{

unsigned char       status;                                  //0x80 - active partition

unsigned char       headStart;                               //starting head

unsigned int        cylSectStart;                  //starting cylinder and sector

unsigned char       type;                                    //partition type

unsigned char       headEnd;                                 //ending head of the partition

unsigned int        cylSectEnd;                              //ending cylinder and sector

unsigned long       firstSector;                   //total sectors between MBR & the first sector of the
partition

unsigned long       sectorsTotal;                  //size of this partition in sectors

};

//Structure to access boot sector data

struct BS_Structure{

unsigned char jumpBoot[3]; //default: 0x009000EB
unsigned char OEMName[8];

unsigned int bytesPerSector; //deafault: 512

unsigned char sectorPerCluster;

unsigned int reservedSectorCount;

unsigned char numberofFATs;

unsigned int rootEntryCount;

unsigned int totalSectors_F16; //must be 0 for FAT32

unsigned char mediaType;

unsigned int FATsize_F16; //must be 0 for FAT32

unsigned int sectorsPerTrack;

unsigned int numberofHeads;

unsigned long hiddenSectors;

unsigned long totalSectors_F32;

unsigned long FATsize_F32; //count of sectors occupied by one FAT

unsigned int extFlags;

unsigned int FSversion; //0x0000 (defines version 0.0)

unsigned long rootCluster; //first cluster of root directory (=2)

unsigned int FSinfo; //sector number of FSinfo structure (=1)

unsigned int BackupBootSector;

unsigned char reserved[12];

unsigned char driveNumber;

unsigned char reserved1;

unsigned char bootSignature;

unsigned long volumeID;

unsigned char volumeLabel[11]; //"NO NAME "

unsigned char fileSystemType[8]; //"FAT32"

unsigned char bootData[420];

unsigned int bootEndSignature; //0xaa55

};

//Structure to access FSinfo sector data struct FSInfo_Structure
```

```
{
unsigned long leadSignature; //0x41615252
unsigned char reserved1[480];
unsigned long structureSignature; //0x61417272
unsigned long freeClusterCount; //initial: 0xffffffff
unsigned long nextFreeCluster; //initial: 0xffffffff
unsigned char reserved2[12];
unsigned long trailSignature; //0xaa550000
};
//Structure to access Directory Entry in the FAT struct dir_Structure{
unsigned char name[11];
unsigned char attrib; //file attributes
unsigned char NTreserved; //always 0
unsigned char timeTenth; //tenths of seconds, set to 0 here
unsigned int createTime; //time file was created
unsigned int createDate; //date file was created
unsigned int lastAccessDate;
unsigned int firstClusterHI; //higher word of the first cluster number
unsigned int writeTime; //time of last write
unsigned int writeDate; //date of last write
unsigned int firstClusterLO; //lower word of the first cluster number
unsigned long fileSize; //size of file in bytes
};
//Attribute definitions for file/directory
#define ATTR_READ_ONLY 0x01
#define ATTR_HIDDEN 0x02
#define ATTR_SYSTEM 0x04
#define ATTR_VOLUME_ID 0x08
#define ATTR_DIRECTORY 0x10
#define ATTR_ARCHIVE 0x20
```

```
#define ATTR_LONG_NAME 0x0f

#define DIR_ENTRY_SIZE 0x32

#define EMPTY 0x00

#define DELETED 0xe5

#define GET 0

#define SET 1

#define READ     0

#define VERIFY 1

#define ADD                  0

#define REMOVE           1

#define LOW                  0

#define HIGH      1

#define TOTAL_FREE 1

#define NEXT_FREE 2

#define GET_LIST 0

#define GET_FILE 1

#define DELETE              2

#define EOF                  0x0fffffff

//************* external variables *************

volatile unsigned long firstDataSector, rootCluster, totalClusters;

volatile unsigned int bytesPerSector, sectorPerCluster, reservedSectorCount;

unsigned long unusedSectors, appendFileSector, appendFileLocation, fileSize, appendStartCluster;

//global flag to keep track of free cluster count updating in FSinfo sector

unsigned char freeClusterCountUpdated;

//************* functions *************

unsigned char getBootSectorData (void);

unsigned long getFirstSector(unsigned long clusterNumber);

unsigned long getSetFreeCluster(unsigned char totOrNext, unsigned char get_set, unsigned long
FSEntry);

struct dir_Structure* findFiles (unsigned char flag, unsigned char *fileName);
```

```
unsigned long getSetNextCluster (unsigned long clusterNumber,unsigned char get_set,unsigned long
clusterEntry);

unsigned char readFile (unsigned char flag, unsigned char *fileName);

unsigned char convertFileName (unsigned char *fileName);

void writeFile (unsigned char *fileName);

void appendFile (void);

unsigned long searchNextFreeCluster (unsigned long startCluster);

void memoryStatistics (void);

void displayMemory (unsigned char flag, unsigned long memory);

void deleteFile (unsigned char *fileName);

void freeMemoryUpdate (unsigned char flag, unsigned long size);

#endif
```

Database header library:

```
#ifndef __DBPK_H__

#define __DBPK_H__

//*************************************************************************

#include "delimfld.h"

#include "len-fld.h"

#include "fix-fld.h"

 class DBPK

 {

        public:

                char username;

                char SHA256s;

                char passclient;

                char Keyclient;

                char ILclient;

                char EPserver;
```

```
                    DBPK();
                    static int InitBuffer(DelimFieldBuffer &);
                    static int InitBuffer(LengthFieldBuffer &);
                    static int InitBuffer(FixedFieldBuffer&);
                    void Clear();
                    int Unpack(IOBuffer &);
                    int Pack(IOBuffer &) const;
                    void Print(ostream &,char *lable=0) const;
                    int writeperson( ostream&,DBPK &);
};
#endif
```

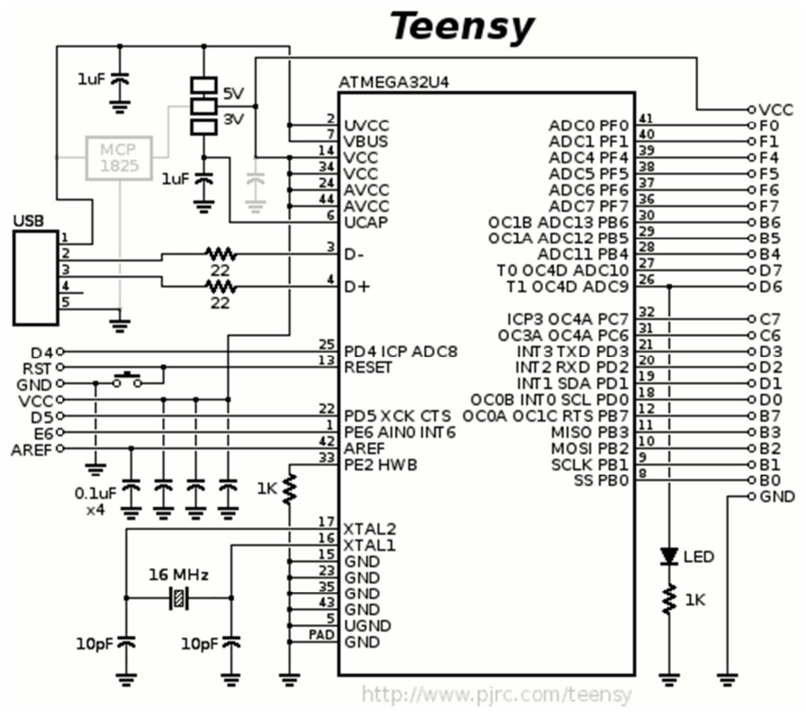
ParseKey+ code:

```
void CreatPK(int x)
{
          int PKFilesize=1024;
          int IL[x];
          srand(time(NULL));
          int RS[x] = rand(Timestamp);
          Key=sha256(RS);
          int PrevIndex=0;
          int PrevSubKeySize=0;
while(sizeof(Key-PrevSubKeySize>0))
{
int Index=(rand()%(PKFilesize-PrevIndex+PrevSubKeySize)+1);
PrevIndex=Index;
          int lenght=(rand()%(sizeof(key-PrevSubKeySize)));
PrevSubKeySize=PrevSubKeySize+lenght;
          for(i=0;i<x;i++)
```

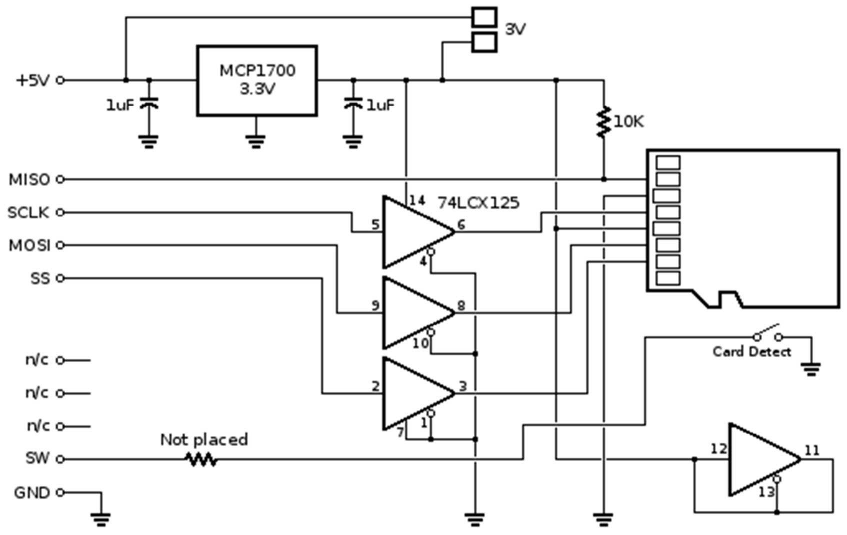
```
             IL[i]=Index[i];

                          for(i=x;i< 2x;i++)

                           for(j=0;j<x;j++)

{

             IL[i]=lenght[j];

}

int parsekey[x]=makePKfile(key[x],IL[x]);

printf("key=%d and IL=%d",key,IL);

}

*******************************************************************

void makePKfile(int key[x],int IL[x])

{

             int i=0;

             int parsekey[x]=0;

             int Hashlength=sizeof(Hash(Timestamp));

             int Iteration=PKFilesize/Hashlength;

                          for (int i=0;i<Iteration;i++)

{

 parsekey[x]= parsekey[x]*sha256(Random seed);

}

while((int current=IL[0])!=0)

{

overwrite();

i++;

IL[0]=IL[i];

}
```

APPENDIX B

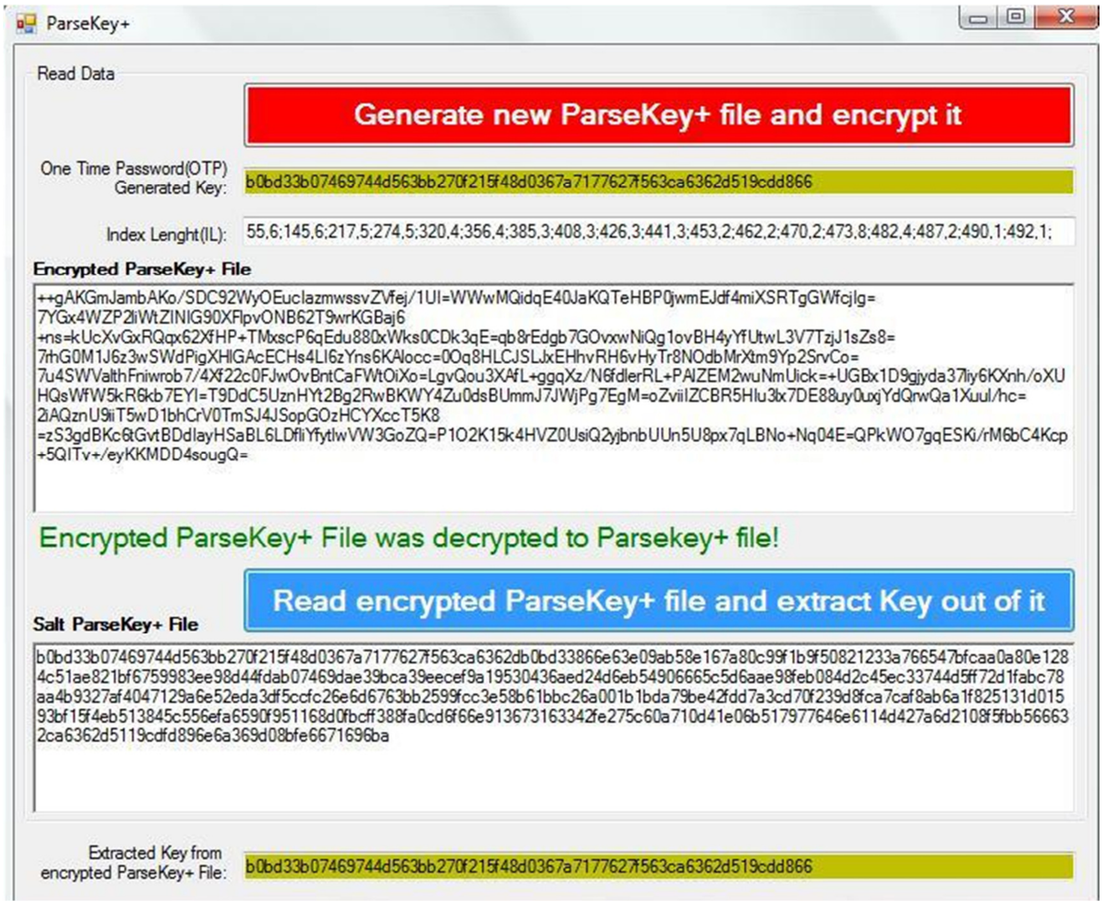
Teensy 2.0 Schematic:



SD Adaptor schematic diagram:



APPENDIX C


ParseKey+
Read Data
Generate new ParseKey+ file and encrypt it
Read encrypted ParseKey+ file and extract Key out of it

One Time Password(OTP) Generated Key: b0bd33b07469744d563bb270f215f48d0367a7177627f563ca6362d519cdd866

Index Lenght(IL): 55,6;145,6;217,5;274,5;320,4;356,4;385,3;408,3;426,3;441,3;453,2;462,2;470,2;473,8;482,4;487,2;490,1;492,1;

Encrypted ParseKey+ File

```
++gAKGmJambAKo/SDC92WyOEuclazmwssvZVfej/1UI=WWwMQidqE40JaKQTeHBP0jwmEJdf4miXSRTgGWfcjIg=
7YGx4WZP2liWtZINIG90XFlpvONB62T9wrKGBaj6
+ns=kUcXvGxRQqx62XfHP+TMxscP6qEdu880xWks0CDk3qE=qb8rEdgb7GOvxwNiQg1ovBH4yYfUtwL3V7TzjJ1sZs8=
7rhG0M1J6z3wSWdPigXHIGAcECHs4LI6zYns6KAlocc=0Oq8HLCJSLJxEHhvRH6vHyTr8NOdbMrXtm9Yp2SrvCo=
7u4SWValthFniwrob7/4Xf22c0FJwOvBntCaFWtOiXo=LgvQou3XAfL+ggqXz/N6fdlerRL+PAlZEM2wuNmUick=+UGBx1D9gjyda37liy6KXnh/oXU
HQsWfW5kR6kb7EYI=T9DdC5UznHYt2Bg2RwBKWY4Zu0dsBUmmJ7JWjPg7EgM=oZviiIZCBR5Hlu3lx7DE88uy0uxjYdQrwQa1Xuul/hc=
2iAQznU9iiT5wD1bhCrV0TmSJ4JSopGOzHCYXccT5K8
=zS3gdBKc6tGvtBDdlayHSaBL6LDfliYfytlwVW3GoZQ=P1O2K15k4HVZ0UsiQ2yjbnbUUn5U8px7qLBNo+Nq04E=QPkWO7gqESKi/rM6bC4Kcp
+5QITv+/eyKKMDD4sougQ=
```

Encrypted ParseKey+ File was decrypted to Parsekey+ file!

Salt ParseKey+ File

```
b0bd33b07469744d563bb270f215f48d0367a7177627f563ca6362db0bd33866e63e09ab58e167a80c99f1b9f50821233a766547bfcaa0a80e128
4c51ae821bf6759983ee98d44fdab07469dae39bca39eecef9a19530436aed24d6eb54906665c5d6aae98feb084d2c45ec33744d5ff72d1fabc78
aa4b9327af4047129a6e52eda3df5ccfc26e6d6763bb2599fcc3e58b61bbc26a001b1bda79be42fdd7a3cd70f239d8fca7caf8ab6a1f825131d015
93bf15f4eb513845c556efa6590f951168d0fbcff388fa0cd6f66e913673163342fe275c60a710d41e06b517977646e6114d427a6d2108f5fbb56663
2ca6362d5119cdfd896e6a369d08bfe6671696ba
```

Extracted Key from encrypted ParseKey+ File: b0bd33b07469744d563bb270f215f48d0367a7177627f563ca6362d519cdd866

ParseKey+ Application Demo