Incorporating Privacy and Security Features in an Open Source Search Engine

Submitted to

Dr. Chris Pollett

By
Akash Patel

1. Introduction

The aim of the CS 297 project is to explore various privacy and a security features of an open source search engine and enhance the security and privacy capabilities of a Yioop. A Yioop is a GPLv3, open source, PHP search engine developed by Dr. Pollett. Since Yioop is an open source it is easy to understand what information it uses from a user request.

Most modern search engines record user IP address, the time of visit and user tracking cookies to keep track of search terms in order to improve efficiency of the search result and show user specific advertisements. This information can be used to reveal personal information and hence many people object to its collection.

For this project, we have implemented the proxy feature and the Perfect Forward Secrecy in Yioop to enhance the privacy one has when using this search engine. We also enhanced Yioop's crawling ability by adding code to crawl the Tor network.

This report includes details of all the deliverables of the CS297 project. Before starting the project, preliminary activity was to install a Yioop on the local apache server. The first deliverable was to write the CS297 proposal. The second deliverable was to prepare a presentation on the Perfect Forward Secrecy. The third deliverable was to write a PHP script to verify whether a website supports the Perfect Forward Secrecy or not. The fourth deliverable was to prepare a presentation on the Tor network and Tor hidden Service, and to write the code to crawl the Tor network. The fifth deliverable was to prepare a presentation on the zero knowledge proof. The sixth deliverable was to add a proxy feature in a Yioop. The last deliverable was to prepare the CS297 report.

In this report, each deliverable is explained under appropriate section headers. At the end, the report has the conclusion section to discuss the learning from the project followed by the reference section including all the references used to achieve the goals of the project.

2. Overview of Deliverables

2.1 Deliverables#1: PPT on the Perfect Forward Secrecy

The purpose of this deliverables is to get an understanding of the Perfect Forward Secrecy (PFS) and prepare a presentation on it.

The Perfect Forward Secrecy (PFS) is a property of the key-agreement protocol that ensures that a session key used to encrypt the data will not be compromised even if in the future, a long term private key is compromised. The idea is not to use a single key (e.g. private key) to generate all the session keys.

In order to understand why do we need to use the Perfect Forward Secrecy that, it is helpful to have a basic idea of how HTTPS works in general. When we access a secure website, it uses HTTPS protocol to make the connections. All HTTPS connections are not equal. Once a connection is established, a browser generates a session key from the private key, and encrypts the data using a session key. An interceptor can intercept all the communication messages. An interceptor does not know the private key of the client (browser), so an interceptor cannot derive the session key. Thus an interceptor cannot decrypt messages, but can still store all the communication messages. Later on, if an interceptor somehow obtains a client's private key, he can derive the session key and decrypt all the stored messages. The main problem here is that, a private key is used for two purposes: authentication and encryption. Authentication only matters while the communication is established, but encryption is expected to last for years. This situation is the motivation factor for the Perfect Forward Secrecy.

In PFS, a client (browser) periodically creates a new session key based on the values supplied by both parties in the exchange. Because both parties contribute a random value known only to them, each new key generated is dissimilar to the previously created keys. Even if a third party managed to intercept a private key, the third party can only derive the session key for a very small duration.

A session key is the key factor to determine whether a connection has Perfect Forward Secrecy or not. The Transport Layer Security (TLS) handshake protocol is a responsible for the authentication and key exchange, necessary to establish a secure connection. To establish a secure connection, TLS protocol needs to do the three things: cipher suite negotiation, authentication of a server and key

exchange. How the session key is derived is determined by the cipher suite in use. In the beginning of SSL handshake, the client sends a list of supported cipher suites. The server then picks one of the cipher suites, based on a ranking and server informs a client which cipher suite will be used from onwards communication. This step determines if future connections will have the Perfect Forward Secrecy or not. The cipher suites that use ephemeral Diffie-Hellman (DHE) or the elliptic-curve variant (ECDHE) have the Perfect Forward Secrecy.

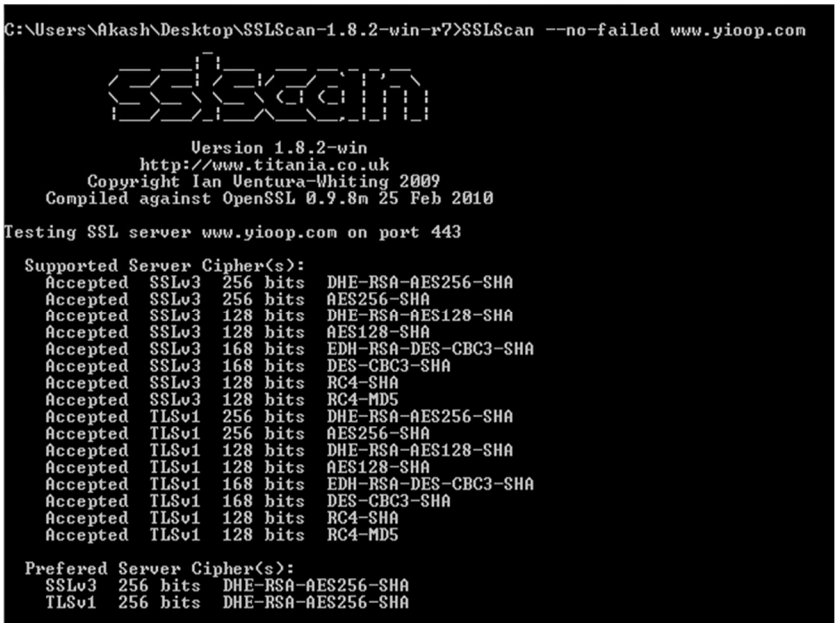
A Web server needs to be configured to use the PFS. A Web server usually has a cipher suite configuration in its SSL configuration. There are two relevant options: first, the cipher suites that you want your server to use, and second, how a server picks the cipher suite. The order of a cipher suite also matters. For the Apache server, Perfect Forward Secrecy requires Apache 2.3.3 or higher. Below examples shows how to configure mod_ssl to enable the Perfect Forward Secrecy.

```
SSLProtocol +TLSv1 +TLSv1.1 +TLSv1.2
SSLHonorCipherOrder On
SSLCipherSuite ECDHE-RSA-RC4-SHA:ECDHE-RSA-AES128-SHA:AES128-SHA:RC4-SHA
```

There are certain challenges for the PFS. The first challenge is that the Diffie-Hellman (DHE) key exchange used in the PFS is significantly slower due to additional calculations of a session key in each request. Web site operators tend to disable DHE suites in order to achieve better performance. The second challenge is from a client side, not all browsers support all the necessary suites.

Deliverables#2: PFS verification and PFS Checker Script

There are many ways to verify whether a web site supports PFS or not. The first is the OpenSSL utility. The second is the SSL scan utility and third is the SSL lab website. I have used the SSL Lab utility to show that a Yioop supports the PFS. A Yioop preferred a cipher DHE_RSA_AES256-SHA for the connection, thus it supports PFS.



PFS Checker PHP Script

PFS checker is a PHP script, used to verify whether a websites support the PFS or not. The Input to the script is an URL of the website and the output of the script is the list of the cipher supported by a website. The snapshot of the output is as shown below.

PFS Support:Yes

Request Header

array(26) { ["url"]=> string(22) "https://www.yioop.com/" ["content_type"]=> string(9) "text/html" ["http_code"]=> int(200) ["header_size"]=> int(547) ["request_size"]=> int(159) ["filetime"]=> int(-1) ["ssl_verify_result"]=> int(19) ["redirect_count"]=> int(0) ["total_time"]=> float(0.093) ["namelookup_time"]=> float(0) ["connect_time"]=> float(0) ["pretransfer_time"]=> float(0) ["size_upload"]=> float(0) ["size_download"]=> float(3954) ["speed_download"]=> float(42516) ["speed_upload"]=> float(0) ["download_content_length"]=> float(3954) ["upload_content_length"]=> float(0) ["starttransfer_time"]=> float(0.093) ["redirect_time"]=> float(0) ["redirect_url"]=> string(0) "" ["primary_ip"]=> string(13) "173.13.143.74" ["certinfo"]=> array(0) { } ["primary_port"]=> int(443) ["local_ip"]=> string(11) "192.168.1.4" ["local_port"]=> int(55871) }

Response Header

Array ([0] => HTTP/1.1 200 OK [1] => Date: Tue, 03 Dec 2013 01:22:39 GMT [2] => Server: Apache/2.2.24 (Unix) DAV/2 PHP/5.4.17 mod_ssl/2.2.24 OpenSSL/0.9.8y [3] => X-Powered-By: PHP/5.4.17 [4] => X-FRAME-OPTIONS: DENY [5] => Set-Cookie: yioopbiscuit=0vopseuhbnferj5ao4kq2j1shl432qpqmrfesom7u9persti2123u7l9cj0kg421bfn0t4mc1m3tpea32abvfch2ae5t5qb1jdlj0t3; path=/ [6] => Expires: Thu, 19 Nov 1981 08:52:00 GMT [7] => Cache-Control: no-store, no-cache, must-revalidate, post-check=0, pre-check=0 [8] => Pragma: no-cache [9] => MS-Author-Via: DAV [10] => Vary: Accept-Encoding [11] => Content-Length: 3954 [12] => Connection: close [13] => Content-Type: text/html)

Deliverables#3: TOR network Presentation

Tor (The onion router) is an open source software used for an anonymous internet surfing. Tor was originally designed, implemented, and deployed as a third-generation onion routing project of the U.S. Naval Research Laboratory. The primary purpose of the Tor was to protect the U.S. navy's confidential communication. Today, it is used by journalists, military people and corporate people for a privacy and security purpose.

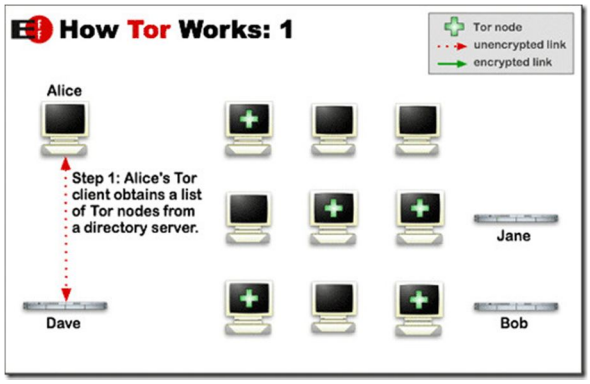
Tor protects against Internet surveillance known as "Traffic analysis". Traffic analysis is a special kind of interference attack used to deduce pattern information from the patterns in a communication. An Internet data packet has two parts: the header and the body. The header has basic information like a source, destination, timing and some geographical information. The body part has an encrypted data. In traffic analysis, an interceptor focuses on the header part and tries to find a common pattern in the header. The header might not reveal the person's exact identity but still reveals much useful information.

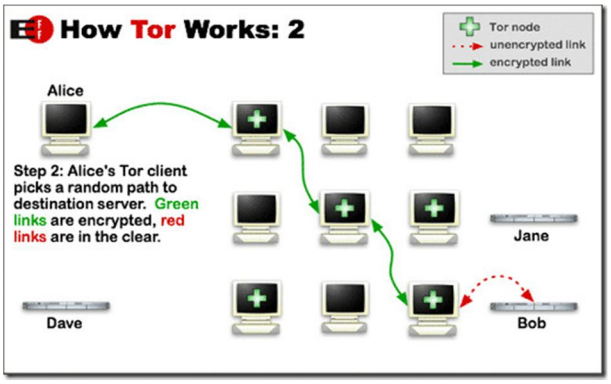
In some countries, there are many restrictions on the internet usage. People cannot share their view and access some of the websites, and if they do, it is very easy for government agencies to identify that person. Tor provides freedom of speech by providing an anonymous internet surfing.

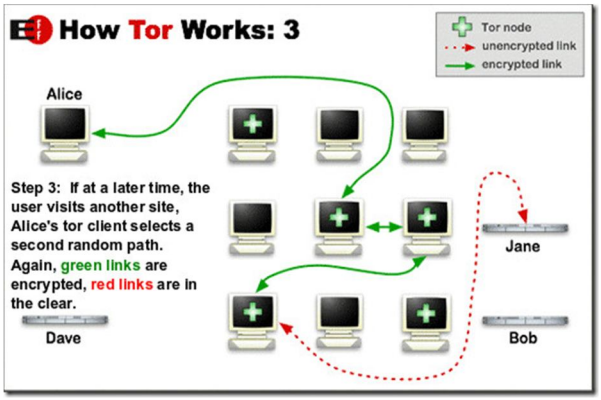
Tor uses an onion routing system. Tor uses thousands of volunteer's networks to direct traffic over the internet, so user's identity can be kept hidden from a network interceptor. Tor helps to reduce the risk of a traffic analysis by distributing transactions over the several places so no single point can link to the sender's destination.

For example, user A wants to send data safely to user B using the Tor network. Tor creates a private network for this communication. The first step is to identify the available nodes. User A's Tor client obtains a list of the Tor nodes from a server. It picks a random node each time so a pattern cannot be observed by an interceptor. A client generates an encrypted message and sends it to the first node. The

Tor client on this node decrypts the first layer of encryption and identifies the next node. This will continue until the final node. The final node receives the location of the actual recipient, where it transmits an unencrypted message to ensure complete anonymity. Now when user 'A' wants to send another packet its Tor client uses a completely different path.







Tor works on the principle of the onion routing. The onion routing uses networking technologies to provide anonymity and privacy on the internet communication. The onion routing connection has three phases: connection setup, data movement and connection tear down. The first phase starts when an initiator creates an onion, a layered data structure that specifies properties of a connection at each point. An initiator determines the number of onion routers (nodes) to be used in the communication and creates the onion packets by having multiple encryptions using a public key of the onion router (node). Each node has information about only two nodes: the sender and the receiver. Each node peels the layer of onion; they use their public key to decrypt the data and obtain information about the next node where data needs to send .Receiver can use its public key and finally obtain plain text. Once a connection is established, bi-directional communication is possible. When data is sent back from the receiver to the sender layering occurs in the reverse direction.

In the onion routing, the setting up a connection and identifying the onion routers for a communication does not take much time. Massive encryption is done at this stage, but since encryption is much cheaper than decryption, more burdens are placed on onion routers. Each onion routers needs to perform decryption which is a costly operation in terms of time. The overall the performance of the Tor network is slow due to extra bouncing of the packets as well as due to the decryption at each of the onion routers. Performance also depends on the bandwidth of each of the onion routers. If one of the routers is slow, then the overall time for communication will increase.

3.1 Tor Hidden Service:

A Tor hidden service allows publishers to publish a service without revealing their identity (IP address). Users can connect to a hidden service using a rendezvous point without knowing the publisher of a service and revealing their own identities. This type of anonymity provides protection against the distributed DoS attacks as an attacker would not know the IP address of a service.

Below are the three design goals of a Tor hidden service.

1. Access Control: A publisher needs a way to filter the incoming requests so that attackers cannot flood a service by making many connections to the service.

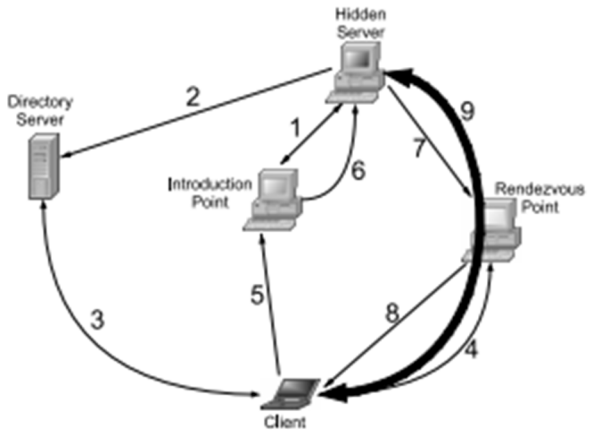
2. Robustness: A publisher should be able to hide his identity for long time, and a service should not be bound with only one onion router. A publisher should allow to migrate a service to the different onion routers.

3. Application transparency: we are forcing user to use a service through the Tor network, but we should not force a publisher to make any changes in the application.

Let's say Bob wants to publish a hidden service. First Bob needs to deploy a service on the server. Then Bob should select the contact point for the service. These contact points are known as introduction points. Bob's Tor client generates the public key and sends the key to the introduction point. Bob's client makes the Tor circuit with the introduction point instead of a direct connection, so Bob's identity is kept private. The Introduction points only receive the public key for the service, not the IP address of a service. A Hidden service assembles a service descriptor which has public key and introduction points, and signs it with the service's private key. This descriptor is uploaded to a distributed hash table. This descriptor can be found by the client by requesting a xyz.onion where xyz is a 16-character name derived from the public key of a service.

Let's say Alice has learned about Bob's service, and she wants to use it. She already knows the descriptor of Bob's service. Alice wants to communicate anonymously. Her Tor client creates the Tor circuit by randomly picking a relay and asking it to act as a rendezvous point. Alice sends a rendezvous cookie to one of the introduction point. The cookie has information about a rendezvous point and an introduction message, encrypted using a service's public key. These communications take place via Tor circuit. A hidden service gets the request and obtains the address of a rendezvous point and sends the one time secret to it. At this point it is very important that it sticks to the same entry guard when creating a new circuit. The entry guard is a set of relays which are always picked as a first node while creating a circuit. These relays are chosen randomly. In the last step, the rendezvous point notifies a client about a successful connection establishment. After that, both a client and service can use their circuits to communicate with each other. A rendezvous point simply relays (end-to-end encrypted) messages from a client to a service and vice versa. Once a rendezvous point gets the response from a service, it informs the client that the connection is established, and now a client and service can talk with each other through a rendezvous point.

In general, six relays are used in the end to end communication. Three of them are chosen by a service (also known as introduction point) and three are chosen by a client including a rendezvous point. This entire process can be depicted by the below figure.



The steps to create a hidden service are as below:

1. Install a web server locally
2. Configure a hidden service to point to the local web server
3. Open the torrc file. It is located at: \Tor\Tor Browser\Data\Tor \torrc

4. Now add an entry like below in the torrc file
    a. HiddenServiceDir D:\Tor\HiddenService
    b. HiddenServicePort 80 127.0.0.1:8080
5. Save the torrc file and restart tor client

HiddenServiceDir is a place where the Tor stores the information about a hidden service. Tor will generate a hostname file in this folder, which has an onion URL for the service. The Hidden service Port is used to specify a virtual port, IP address, and port for redirecting connections to this virtual port.

3.2 Code to crawl the Tor network

I have added the code in the functions: getPages, prepareUrlHeaders, and getPage of fetch_url.php of Yioop. I have also created the patch of changes and uploaded it on the issue tracker.

Deliverables#4: Proxy server presentation and proxy feature in Yioop:

In the client-server communication, a proxy server acts as an intermediate machine between a client and server. A proxy server is a substitute for connecting directly to the web. A client connects with a proxy server and sends the request to a proxy server, then proxy server forwards that request to a web server. A web server sends a response to a proxy server and it directs the response to a client. A proxy was invented to add the structure and encapsulation to the distributed systems.

There are many security advantages of a proxy server. It hides the internal clients from an external network, blocks dangerous URLs, filters a dangerous content, checks the consistency of a retrieved content, and eliminates the need for a transport layer routing between the networks. Proxy server also provides a single point of access, control, and logging.

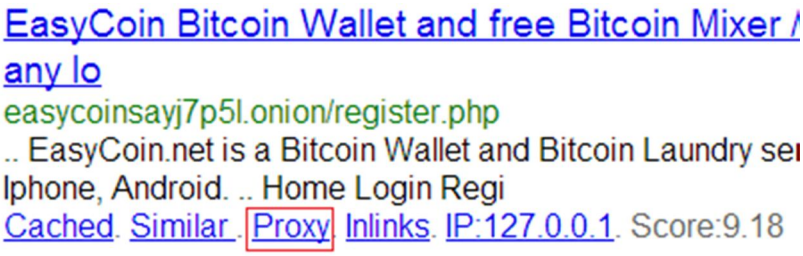
Forward proxies, open proxies, reverse proxies and performance enhancing proxies are the common type of a proxy.

A Forward proxy acts as a gateway for a client browser. It gets the HTTP request from a client and forwards it to a Server thus provides anonymity to a client IP.

4.1 Proxy feature in Yioop:

I have added the code to create a proxy link for an each result link. When a user clicks on a proxy link, a request is sent to a Yioop server, and it sends a request to the requested website. In this case, a Yioop server acts as a forward proxy server. A Yioop server uses the Tor connection to make a request to the website, so its identity can be kept private.

I have modified the search_view.php to add the code to create a dynamic proxy link. I have also added the proxy_url.php file to receive a requested URL by user and to make a request to the URL through the Tor network. The snapshot of a proxy link is as shown below.



Deliverables#5: Presentation on the Zero Knowledge Proof:

A zero knowledge proof or protocol is a method in which a party A can prove that given statement X is certainly true to party B, without revealing any additional information of a statement X.

Let's say Alice and Bob want to communicate over a shared network. Alice initiates the communication and sends a secret to Bob. Bob verifies a secret, so he can be certain that he is communicating with Alice. Once he verifies the secret, he sends a conformation. In the above scenario, Bob must know Alice's secret, so he can verify Alice's identity but now Bob can impersonate Alice. The zero knowledge proof protocol allows Alice to prove Bob that she knows the secret without revealing the secret. In this protocol, verification is performed for many executions and each time, Alice needs to pass the verification.

A zero knowledge protocol is a three pass identification protocol. The first message is a commitment or witness, sent from Alice to Bob. The second message is a challenge sent from Bob to Alice and the third message is a response sent from Alice to Bob.

A zero knowledge protocol must have below three properties.

1. Completeness: If the statement is true, an honest verifier will be convinced by an honest prover.

2. Soundness: If the statement is false, an interceptor cannot convince a verifier that it is true, except with small probability.

3. Zero-knowledge: If the statement is true, no cheating verifier learns anything other than this fact. The randomness is also an important property of the zero knowledge protocol. The randomness in the commitment and challenge message are used to hide the secret information.

Fiat-Shamir protocol is an example of Zero Knowledge Proof protocol. The goal of this protocol is to prove that prover knows the secret in n executions. This is a probabilistic protocol with the probability of a 2^{-n} for adversary to fool a verifier. Usually the numbers of executions are around 20 to 40.

This is a secure protocol. The prover does not need to share the secret. It is also simple, as it does not involve any complex encryption method. There are certain limitations of this protocol. First the secret must be numeric, otherwise a translation is needed and second, it is only suitable for a certain kind of problem like the grandmaster problem and the mafia problem.

3. Conclusion

The deliverables in my CS297 project helped me to understand the different security concepts. Deliverable-2 helped me to understand the Perfect Forward Secrecy (PFS) concept and write the PFS verification script, which shows that a Yioop supports the Perfect Forward Secrecy. Deliverable-3 gave me an opportunity to understand the Tor network, Tor hidden service and helped me to write the code to crawl the Tor network.Deliverable-4 gave me an opportunity to learn about zero knowledge proof protocol. Deliverable-5 helped me to learn about a proxy feature.

4. References

1. http://en.wikipedia.org/wiki/Perfect_forward_secrecy

2. Section 9.3.4 of Information Security Principles and practice by Dr. Mark Stamp, Published by JohnWiley & Sons, Inc

3. http://tinyurl.com/lamysw6

4. Section 9.3.4 of Information Security Principles and practice by Dr, Mark Stamp 5. http://crypto.stackexchange.com/questions/8933/how-can-i-use-ssl-tls-with-perfectforward-secrecy

6. https://scottlinux.com/2013/06/26/how-to-enable-perfect-forward-secrecy-in-apacheon-linux/

7.https://community.qualys.com/blogs/securitylabs/2013/06/25/ssl-labs-deployingforward-secrecy

8. http://en.wikipedia.org/wiki/Tor_(anonymity_network)

9. https://www.torproject.org/about/overview.html.en

10. Technical paper: Onion Routing for Anonymous and Private Internet Connections by David Goldschlag Michael Reedy Paul Syversony on January 28, 1999

11. Technical paper: Onion Routing Efficiency for Web Anonymization in Various Configurations by Tomas Sochor University of Ostrava, Ostrava, Czech Republic

12. https://www.torproject.org/docs/hidden-services.html.en

13. Technical Paper: Section 5 of Tor: The Second-Generation Onion Router by Roger Dingledine, Nick Mathewson and Paul Syverson

14. Technical Paper: Section 5.1 of Tor: The Second-Generation Onion Router by Roger Dingledine, Nick Mathewson and Paul Syverson.

15. Technical Paper: Figure 1 of Locating Hidden Servers by Lasse Øverlier and Paul Syverson

16. https://www.torproject.org/docs/tor-hidden-service.html.en

17. http://en.wikipedia.org/wiki/Zero-knowledge_proof

18. Section 9.5 of Information Security Principles and practice by Dr. Mark Stamp

, Published by JohnWiley & Sons, Inc

19. Technical Paper: How to Explain Zero-Knowledge Protocols to Your Children.

Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings