Trusted Puzzle Solvers without Clock Support against Spam and Denial-of-Service Attacks

Rui Deng
Jiangnan Institute of Computing Technology, Wuxi, China
Email: drui118@163.com

Zuoning Chen
Jiangnan Institute of Computing Technology, Wuxi, China
Email: chenzuoning@163.net

***Abstract*—Spam and Denial-of-Service (DoS) attacks are growing threats on the Internet. Cryptographic puzzles can be used to address the problem effectively. Additionally, the Trusted Puzzle Solver (TPS), which is similar to the Trusted Platform Module (TPM), is also proposed to help constructing crypto puzzles that may have some outstanding specialties in security and efficiency. Based on the analysis and discussion of two existing crypto puzzles relying on the TPS with a trusted clock, two new improved crypto puzzles are presented, and all the drawbacks found in the old ones are eliminated. The TPS in the new crypto puzzles doesn't have to be equipped with a clock thus can be realized by the TPM without modification. Also, prototype experiments show that the new crypto puzzles can do better in mitigating DoS attacks.**

***Index Terms*—TPM, DoS, spam, cryptographic puzzle, Trusted Puzzle Solver**

I. Introduction

Spam[1], or unsolicited email, is a big threat to email systems. Attackers can flood users' mailboxes with low costs, while taking down a web server within a short period of time, resulting in the loss of time and money to both users and service providers. And also, Denial-of-Service (DoS) attack that can exhaust a server's resources easily and harmfully is a growing concern on the Internet and other open communications systems. A lot of research has been done in the past years, aiming to seek effective solutions to these problems.

Cryptographic puzzles, originally proposed for securing key agreement[2], were first proposed for fighting spam by Dwork and Naor[3], and have been widely used now in mitigating spam and DoS. Proof-of-Work puzzles and Time-Lock Puzzles are two main types of crypto puzzle models in use.

Proof-of-Work puzzles[4], also known as client puzzles[5-6], impose costs on the clients by forcing them to do some work (i.e. crypto calculations) per service access, thereby consuming their resources and slowing down the service request rate. However, it should be noted that client puzzles' generation, as well as the verification of their solutions must be done efficiently, otherwise these two operations would become new DoS attack surfaces. It's also very common that some service providers prepare enough puzzles and corresponding solutions in advance, and don't generate or solve puzzles online when providing services. In this way, the latency of generating and solving the puzzles are hidden from the users when they are trying to access services. However, the adversary can accumulate enough puzzles and solutions and use all of them at the same time to launch DoS attacks. Examples of exploring Proof-of-Work puzzles include using pre-challenges (the pre-challenges can range from security questions to micro-payments) against spam[7], "hashcash"[8] invented by A.Back, defending against (distributed) DoS attacks[9-14] and Sybil attacks[15-16], metering visits to websites[17], providing incentives in P2P systems[18], and rate-limiting TCP connections[19].

The researchers in the study of utilizing crypto puzzles against DDoS attacks concentrate mainly on constructing puzzles that can't be solved in parallel. Q. Tang[11] proposed RSW client puzzle scheme, which is based on the repeated squaring technique therefor is parallel computation resistant. They also proposed two batch verification modes for the RSW client puzzle scheme in order to improve the verification efficiency of the server, and described how to integrate the scheme with reputation systems to further improve the effectiveness in practice. Douglas[20] further improved security definitions for client puzzles, and proposed that solving n puzzles should be n times harder than solving one puzzle.

A Sybil attack[21] consists of an attacker introducing a large number of phantom nodes into a network. Without centralized admission control, it's difficult to distinguish multiple nodes operated by a single attacker from several independent nodes. Since the cost of participation in a p2p network is usually low, resourceful attackers can introduce enough phantom nodes so that they can control a very large fraction of all nodes. This can be used for denial of service or other abuse of the network. N. Borisov[15] proposed a fully decentralized scheme in which locally generated challenges are continually distributed and then incorporated into the puzzle solutions to ensure the freshness of the puzzles. Therefor puzzle solutions can't be reused over time by attackers.

doi:10.4304/jsw.9.10.2538-2545

Meanwhile, Li F[16] presented the SybilControl scheme for controlling the extent of Sybil attacks. It's an admission and retainment control scheme that requires the nodes in a distributed system to periodically solve computational puzzles and prevents the malicious influence of misbehaving nodes that do not perform the computation work.

Time-Lock puzzles, which were formally proposed by Rivest et al.[22], have the property that it can be solved only after a predetermined time. They can be used to send information to the future, i.e., encrypting a message so that no one can decrypt it until the predetermined time. Good Time-Lock puzzles have the property that the algorithm of solving them can't be parallelized. That means multiple machines won't be any faster than a single machine in solving a Time-Lock puzzle. Applications of Time-Lock puzzles include timed release from bilinear pairings[23], non-interactive timed-release[24], proofs of sequential work[25], timed-release encryption scheme that provides user anonymity[26-27], timed release of digital signature[28] and commitments[29], time capsule signature[30], and offline submission[31]. However, Time-Lock puzzles can also be used in fighting spam and DoS attacks, as long as the secret in the puzzles must be checked by the server before the client can access the service. As a result, the clients are forced to be delayed for a period of time before getting served.

Various techniques, such as trusted computing, have been utilized in constructing crypto puzzles recently. The core of trusted computing is the Trusted Platform Module (TPM)[32], which is basically a crypto-enabled smartcard soldered on the mainboard of a PC. With the help of a set of platform configuration registers (PCRs), the TPM participates in the booting process to help building trust from BIOS to OS kernel and finally to the application programs. And it can further prove to remote parties what the current platform configuration is, which is named "attestation". Using PCRs, the TPM can indicate the hardware and software configuration of the platform, and releases secrets to the platform only if the PCRs show the right value, which is termed "unsealing" or "unwrapping". The TPM can also create and store cryptographic keys, either symmetric or asymmetric. One of the keys is a non-migratable signing key, the Attestation Identity Key (AIK), which can uniquely identify the TPM. An AIK could be verified by a third party (i.e. a Privacy-CA) that creates a certificate for it. This certificate, also known as identity credential, is sent to the TPM and later used in attestation. The TPM is embedded in cryptographic algorithm engines such as RSA, HMAC and SHA-1 to support the functions discussed above, but the computational capability is very poor and improper for multiple low latency scenarios, due to the architecture and hardware restrictions.

Patrick and Sean[33] proposed the use of Trusted Puzzle Solvers (TPS) which could be realized by using TPM in constructing Proof-of-Work puzzles and Time-Lock puzzles. However, the TPM has to be altered to be equipped with a trusted clock to adapt to the proposed TPS and crypto puzzles. The trusted clock in the TPS plays an important role in the presented puzzles. The proposed TPS has the following advantages over traditional methods. It saves computing resources and/or time as the puzzles would be solved by TPS. And the time it takes to solve puzzles may vary dramatically across different computing platforms in traditional methods, while can be uniform under TPS.

Our contribution in this paper can be summarized as follows. Through an analysis of the crypto puzzles proposed in Ref.[33], we point out that these crypto puzzles can't defend against DoS attacks well under some conditions. And further, two new improved crypto puzzles are proposed, in which the TPS used doesn't need a trusted clock, thus can be realized by using TPM without any modification. The timing work is done uniformly on the server without clients' participation, therefor the QoS (Quality of Service)[34] can be regulated well by the server. A prototype experiment shows that the new crypto puzzles can do better in mitigating DoS attacks.

The rest of the paper is organized as follows. In section II, crypto puzzles proposed in Ref.[33] are reviewed and analyzed, and their drawbacks are discussed. And then, new improved crypto-puzzles for TPS without trusted clock are proposed in section III. And a following discussion and comparison between previous puzzles and new ones are presented in section IV. Prototype evaluation is described in section V before we draw the conclusions in section VI.

II. Previous Crypto Puzzles And TPS

The TPS proposed by Patrick and Sean[33] is analogical to the TPM in conformation and function, except for a trusted clock. All client machines are required to be equipped with a TPS to perform the puzzle protocols. Every TPS has a distinct asymmetric key pair (a private key *sk* and a public key *pk*) which is similar to AIK in the TPM, a cryptographically secure random number generator (RNG), a cryptographic engine to perform operations such as HMAC, digital signature and public-key decryption, and a trusted clock. All these requirements are fulfilled by the TPM except for the trusted clock that is not available inside TPM. The clocks in the TPSs are key elements in the proposed puzzle protocols. But the clocks need not be synchronized to a global clock and may be reset at power-on, as long as they are all ticking at the same and reasonably precise frequency. Based on the TPS, two crypto puzzle models were presented, which will be reviewed, and discussed below.

A. Proof-of-Work Puzzles

Proof-of-Work puzzles are used to rate-limit service accesses. This model relies on the TPS of the client to solve the puzzles, and only the TPS knows how to solve them up to a certain rate. A puzzle in this model consists of a nonce and a *fee*. The nonce prevents the clients from replaying puzzle solutions, while the *fee* is a parameter used by the trusted clock in TPS to decide the time taken

to solve the puzzle. To make sure the TPS doesn't solve the puzzles too quickly, an inside tick counter register named *balance* is increased periodically (increased by 1 every millisecond in this model). Given a puzzle with a *fee*, the TPS solves it only if the current *balance* is no less than the *fee*. A puzzle in nature is a signature request on the nonce by the TPS, as only the TPS know the *sk*, no one can forge the signature. Thus together with the *fee*, only the TPS can control the puzzle answer and the time taken.

The puzzle protocol executed between a server and a client is depicted as in Fig. 1. We put ***C*** for Client and ***S*** for Server.

1) ***C*** sends a request for service to ***S***.

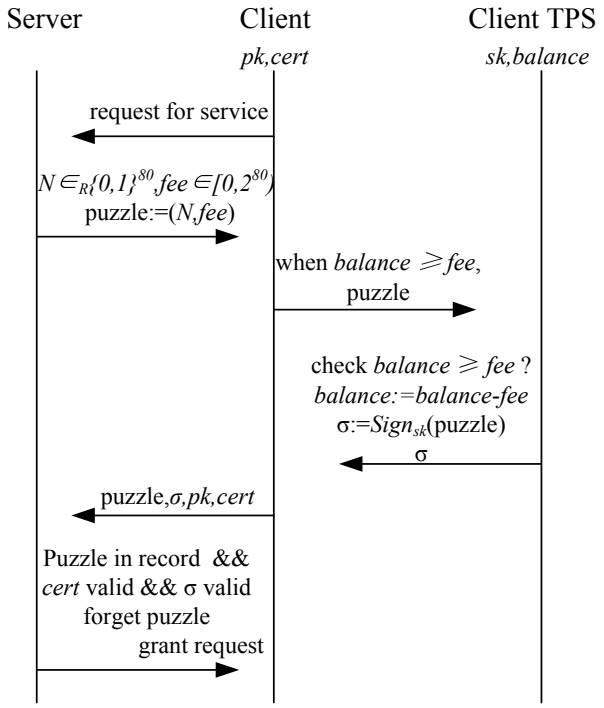

Figure 1. Protocol Description of Proof-of-Work Puzzles

2) ***S*** records and returns the puzzle=*{N,fee}* to ***C***, where *N* is a 80-bit random nonce and fee is a 80-bit non-negative integer.

3) ***C*** waits until *balance* ≥ *fee,* and sends the puzzle to TPS to get the signature. On getting the puzzle, the TPS first checks *balance* ≥ *fee*, if true, signs the nonce in the puzzle and decrease *balance* by *fee*. ***C*** sends the signature (σ) together with the *pk, cert* and the puzzle back to ***S***.

4) ***S*** checks whether the puzzle is in record, then whether the *cert* is valid, and finally verifies the signature on the nonce in the puzzle using the *pk*. Otherwise, he declines the request.

This model uses the TPS to solve the puzzle and no other resources are required, so the client machine is free from the puzzle work and the whole efficiency gets improved. Also the TPS is tamper-proof, the security of the puzzle protocol is guaranteed. This model might work well in mitigating spam and DoS most of the time, but we find out that it may not function well under the following conditions.

Note that when a puzzle is received, a *good* client may query the *balance* and waits until *balance* ≥ *fee* before he asks the TPS to solve the puzzle. But an *evil* client may take it as a chance to start DoS attacks. He doesn't care about whether or not the signature on the nonce can be verified, so a random number may be sent to the server as σ. And also he doesn't bother TPS for σ, and the *fee* restriction is also ignored. Moreover, the *evil* client can always get the right *pk* and *cert* which are often public and easy to obtain. Once he gets the puzzle, the evil client immediately forges an answer and sends it back to the server, while the TPS is bypassed. Back in the sever end, because the *cert* is always valid, it's until the last verification step being done that σ gets checked, before the server can make the final judgment on the puzzle answer. But at that time, all the expensive crypto operations have been done, and the precious computation resources are wasted for nothing, while the *evil* client spends nothing in this attack and also breaks the time limit restriction. Attackers might break down a server easily in a short time using this kind of DoS bomb. It's also observed that the trusted clock in the TPS used for enforcing the *fee* policy doesn't help in slowing down the attack, and the TPS itself gets bypassed. This model might work well against spammers who do have to get the service to accomplish the attack, but it just can't defend against the DoS attack discussed here.

And also, for the *fee* policy, it may not be an easy task for the server to decide the *fee* for the puzzles. One way is to decide the *fee* according to the server's status, the busier the larger, and the freer the smaller. But this strategy doesn't take clients' *balance* into consideration. Assume the server is very busy, he increases the *fee* to a large number to relieve the pressure and to protect the QoS, but some new clients' *balance* might have been increased to a number much larger than *fee*, these new service requests will get fulfilled immediately, and the policy fails. And there are also times when the server gets freer, but the *fee* is still too large for some clients' *balance*, resulting in unnecessary service delay and resource waste.

To ensure that multiple applications on the same machine can solve their own puzzles in parallel, maintaining a separate *balance* for each application was also talked about in this model. These new *balance* registers were proposed to be saved outside the TPS, and protected by techniques similar to "sealing" in the TPM. We think it might not be practical to do so. Considering that these *balance* registers are frequently accessed and modified (by 1 per millisecond), techniques like "sealing" are too complicated and time-consuming, not to mention the poor processing speed of the TPM.

All the flaws discussed above would be fixed in section III where the new improved puzzles are proposed.

B. Time-Lock Puzzles

This puzzle model also relies on the TPS present in the client machine as a trusted time server. The TPS makes sure that sufficient time has elapsed before the puzzle is solved. And also the TPS must be the only possible puzzle solver for the client. Besides timed release

applications, this puzzle model can also be used for mitigating spam and DoS attacks. The puzzle protocol is depicted as in Fig. 2. We put ***C*** for Client and ***S*** for Server.

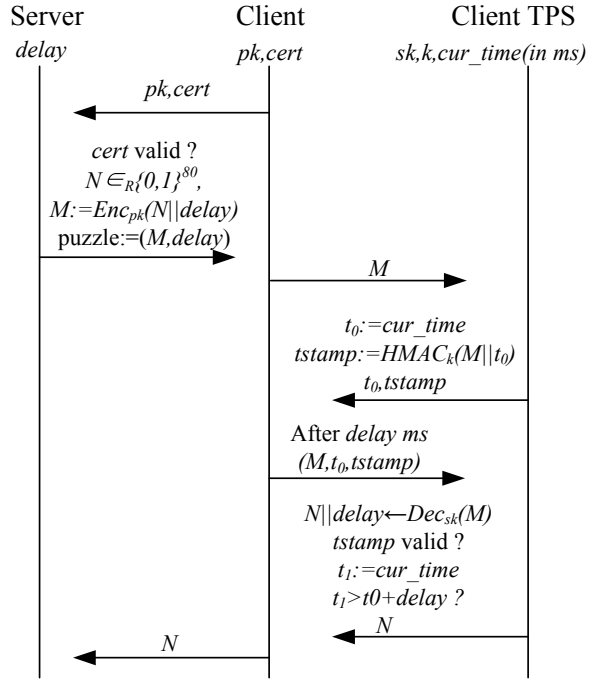

Figure 2. Protocol Description of Time-Lock Puzzles

1) ***C*** sends *pk* and *cert* of his TPS to ***S*** for a Time-Lock puzzle.

2) ***S*** checks if *cert* is a valid certificate for *pk*. On success, he returns ***C*** with *{M ,delay}* as the puzzle, where *delay* is a 80-bit non-negative integer that denotes the time (in ms) necessary for solving the puzzle and *M* is the encryption of a 80-bit random nonce *N* concatenated with *delay* under *pk* .

3) ***C*** relays ***M*** to his TPS. Let the current time be t_0. The TPS returns t_0 and *tstamp* to ***C***, where *tstamp* is time-stamp on *M* at time t_0 under the TPS's secret HMAC key *k*, i.e. $tstamp=\text{HMAC}_k(M||t_0)$.

4) After *delay* ms or more, ***C*** comes back and relays *{M,t0,tstamp}* to the TPS. The TPS first decrypts *M* to *N*||*delay* using *sk*, and then checks $tstamp=\text{HMAC}_k(M||t_0)$, and finally $t_1 \geq t_0+delay$,where t_1 is the current time. The TPS returns *N* on all the checking passed or an error to ***C***.

5) ***C*** sends *N* to ***S*** who will validate *N* and finally fulfilled the service request.

This puzzle model does work well for the *good* client, and can counter spammers who have to solve the puzzle first before the attacks take effect. But similarly, it still might not be able to mitigate the DoS attacks under certain conditions.

For the same reason discussed, the DoS attacker can just ignore the *delay* requirement and bypass the TPS. He can flood the server with wrong *N*. What is worse is that in step 1 and step 2, the attacker spends nothing to get a puzzle that is generated by the server with two expensive cryptographic operations, *cert* validation and *pk* encryption. The attacker may only need to flood the server with step 1 and step 2 to accomplish the attack, without even bothering to send forged *N*. The problem here is that it's just too easy for an attacker to send a puzzle request which costs the server a relatively considerable amount of computational resources.

In the next section, two improved versions of the puzzle models discussed above are presented and all the shortcomings described above are eliminated.

III. New Crypto Puzzles and TPS without Clock Support

It's observed from the analysis in last section that the time-limit or time-delay policy can't be enforced faithfully on the client machine even the clock on the TPS is trustworthy. So it is improper to rely on the client to do the timing work.

In the new puzzle model, the timing work is moved to the server end. Since the trusted clock on the TPS is only used for time service in the old models, the TPS used in new puzzle models needn't be equipped with a clock anymore. Hence the TPS can be totally compatible with the TPM available on many PCs, and can be realized by the TPM without modification. The new models only use the TPS as a non-substitutable puzzle solver to every client. For the convenience of discussion, the same terms and symbols in the old puzzle models are used in the new improved ones.

A. New Proof-of-Work Puzzles

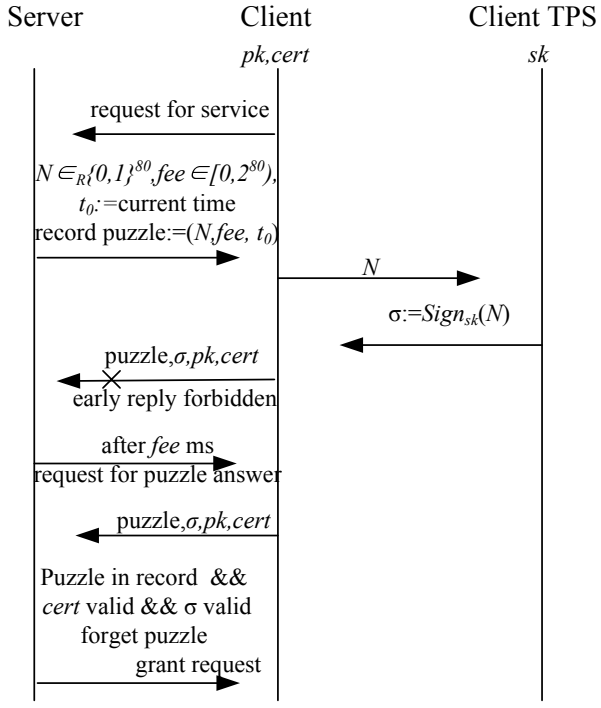

Figure 3. Protocol Description of New Proof-of-Work Puzzles

In this new model, the timing work is done by the server itself and every puzzle can only be tried once. The puzzle protocol is depicted as in Fig. 3.

1) ***C*** sends a request for service to ***S***.

2) ***S*** records*{N,fee,t0}* and returns the puzzle=*{N, fee}* to ***C***, where *N* is a 80-bit random nonce ,*fee* is a 80-bit non-negative integer which represents the time delay

before the puzzle answer is required, and t_0 is the current time.

3) ***C*** sends the puzzle to TPS to get the signature. On getting the puzzle, the TPS signs the nonce and returns the result back to ***C***. ***C*** doesn't use the *fee* in the protocol as the timing job isn't done here. But the *fee* is still included in the puzzle, for the client might need it. For example, the client should be told how long it takes approximately before he can get serviced, so he can rearrange his schedule accordingly. And also, during this time interval, the client machine can run any other tasks, rather than busy-waiting or stalling.

4) When the time for a puzzle is due, ***S*** sends a request for the puzzle answer to ***C*** and waits for the reply for a certain period of time. If he doesn't get the reply before the timeout, the protocol is aborted. Note also, any *early arriving* reply, that means a surprise reply from ***C*** before ***S*** initiates the request, would be deemed an attack attempt, and the protocol is aborted. On receiving the puzzle answer, ***S*** checks whether the puzzle is in record, then if the *cert* is valid, and finally verifies the signature on the nonce in the puzzle using the *pk*. Otherwise, he declines the request.

The new Proof-of-Work Puzzles model is free from the drawbacks in the old one. First, the timing work is done on the server side, so the TPS doesn't have to embed a trusted clock inside. The requirement for the TPS is lowered. Also, the DoS attack described in section II can't be applied here, because any early arriving reply can be identified easily and is considered as an attack, therefore ***C*** can only send back the reply when he gets the request from ***S***. Thus the time policy can be enforced faithfully. Even if the attacker bypasses the TPS and sends back a wrong answer, ***S*** can just find it out and simply aborts the protocol as he does in other unsuccessful protocol executing instances.

It's also noticeable that ***C*** doesn't have to maintain a unique *balance* register for every separate application to support parallel processing. As for the *fee*, the server can simply adjust it only according to the service status. For example, every *fee* in the puzzle can be calculated based on the amount of unsolved puzzles, or the requests waiting to be fulfilled.

B. New Time-Lock Puzzles

The new Time-Lock-Puzzles also place the timing work on the server side to protect the time delay in each puzzle. But the DoS attack described in section II still needs to be addressed first before the new puzzle protocol is presented. As pointed out in section II, it's much more expensive to generate a puzzle by ***S*** than to generate a puzzle request by ***C***. So it might be proper to impose a comparative computation on ***C*** before ***S*** generates a puzzle. That forces the client to commit its computational resources to the protocol run before the server allocates its memory and processing time. There are various ways to do it, and we choose the method in Ref.[5] to use in the new puzzle model as an example. Any other techniques, such as the one in Ref.[10], that can achieve the same goal are feasible in this model.

It's also noticeable that it's not necessary to perform an expensive cryptographic operation to verify the *cert* for *pk* every time; a cashed lookup table may help. When S gets *{cert, pk}*, he first checks the lookup table, if hit, no further validation is needed. Otherwise, the normal verification is performed, and on success, the new item is added to the lookup table. And old items in the table can be removed according to its last hit time.

The puzzle protocol is depicted as in Fig. 4 and every puzzle can only be tried once.

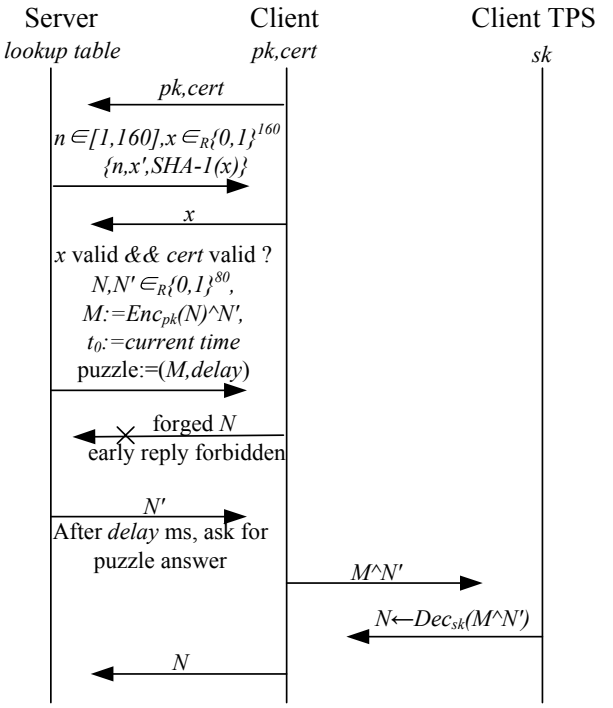

Figure 4. Protocol Description of New Time-Lock Puzzles

1) ***C*** sends *pk* and *cert* of his TPS to ***S*** who records them temporarily.

2) ***S*** sends *{n,x',SHA-1(x)}* to ***C***, where *x* is a 160 bits nonce, and *x'* is *x* with its *n* lowest bits set to 0. ***C*** has to try all the possible values to regenerate *x*. This should take, on average, 2^{n-1} calculations of SHA-1. After that, ***C*** sends *x* back to ***S***.

3) ***S*** waits for the result with a timeout that varies according to *n*. If the received *x* is incorrect, the protocol is aborted. Otherwise, *S* searches for *{pk,cert}* in the lookup table. If hit, he generates the puzzle. If missed, he does the normal verification and then generates the puzzle for ***C***. *N* and *N'* are 80 bits nonce, and the low 80 bits of $Enc_{pk}(N)$ are masked by *N'*. Note that the current time t_0 when the puzzle is generated is also saved to facilitate the timing work.

4) On receiving *M*, ***C*** can't relay it immediately to the TPS, for he also needs *N'* to unmask *M*. The *delay* is also sent to the client. During the *delay* ***C*** is free to do anything meaningful until he gets the answer request from ***S*** at the end.

5) When the time for a puzzle is due, ***S*** sends a request for the puzzle answer together with *N'* to ***C*** and waits for the reply for a certain period of time. If he doesn't get the reply before the timeout, the protocol is aborted. With *N'*,

C can unmask *M* and get the correct *N* from his TPS. Note also, any early arriving reply would be deemed an attack attempt, and the protocol is aborted. ***S*** fulfills the service request if the received *N* is valid.

The time delay is the most important thing in the Time-Lock Puzzles. In this model, the time delay is ensured by the server end, thus the attacker can't bypass the time policy from the client end. And also, to counter the DoS attack in the initial 2 steps in the old puzzle model, a SHA-1 puzzle is used to slow down the clients. If the attacker just ignores the SHA-1 puzzle and forges a wrong answer, the server can find it out effortlessly and abort the protocol. It's also observed that the Parameter *n* in the SHA-1 puzzle can be used to regulate the clients so that everyone is served according to the service policy. For example, to ensure that everyone gets equal service, the *n* is larger for clients that initiate more requests than for those that commit fewer requests. The lookup table used in the protocol to save cryptographic calculations can also help to save the requests' information from the same client, such as requests per minute, which can be used in determining *n* for separate clients.

IV. DISSCUSSION

From the description in last section we can see that the two new puzzle models don't have the flaws found in the old ones, and the trusted clock is also removed from the TPS. As the timing is uniformly processed by the server, the client only needs the TPS to solve the puzzles. Thus the new puzzle protocols are simpler and more efficient than the old ones, while remains the same security. While the clients are free from trusted clocks, the server in the new models needs to do the timing work which is easy to implement on the server machine. For example, a thread in the service process waking up every 10 ms and queuing up all the time-due requests into other service-working threads might be a good choice. And finally the new models can work as well as the old ones, but can counter DoS attacks that overwhelm the old models.

Without trusted clock, the TPS can be realized by TPM without modification. The AIK that can uniquely identify the TPM is a good candidate for *{sk,pk}* of the TPS. But the AIK can only be used as a signing key; it could not be used in Time-Lock-puzzles. Keys generated by TPM and used for RSA encryption and certified by AIK can be used instead.

It should also be noted that in step 1 and step 2 of both new and old Proof-of-Work puzzles, ***C*** spends nothing to get the puzzle, though it's also cheap for ***S*** to generate a puzzle. But ***S*** has to remember every puzzle he generates until he can safely delete it, i.e., ***C*** doesn't send back the puzzle answer after a certain period of time, or ***C*** does send back the puzzle answer and passes the verification, or fails the verification (depending on how many times a puzzle can be tried). Anyway, ***S*** has to save the puzzles for a while. If malicious users flood the server with many requests without answering it, the server would be in trouble of maintaining a large number of unsolved puzzles which will never get processed by *evil* clients. It's suggested that audit be used to partly solve the problem. And methods used in the new Time-Lock-puzzles may also be of some help.

The SHA-1 puzzle used in the new Time-Lock-puzzles may fail in fighting DDoS attackers who have much more computational resources than the server. And it is even worse that the SHA-1 puzzle can be solved in parallel. In that case, the Repeated-Squaring method[10][35]that is non-parallelizable might be used instead.

V. PROTOTYPE EVALUATION

The service latency increase under DDoS attack is evaluated in both old and new models. The experiment scenario is set as below. 8 computers, each equipped with a 3GHz Core 2 processor, 4GB RAM, Linux Redhat 5 OS, are configured as the servers. While as many as 80 other computers are used to simulate clients in the experiment, and all of them are comparable to the server in computational ability. And in every client computer, at most 1000 threads are used to simulate working clients, *good* or *evil*. So there are at most 80000 simulated clients in the scenario. All the computers are connected in the local Gigabit network. The clients connect to the 8 servers randomly.

Since there is no TPM with trusted clock available, and the TPM is not suitable to serve 1000 threads in RSA computation, the puzzle protocols have to be predigested before evaluation. Firstly, the *delay* or *fee* is set to 0, so no clock is needed in TPS in the old model, and also the *delay* is always neglected by attackers. Secondly, we use 1024-bit RSA signature as the puzzle, and for simplicity, no cert is used, but *pk* is used to identify the client. And lastly, TPS is simulated by software (as a dynamic linked library simulating all the needed functions in the real TPS, such as crypto operations), and all the RSA and other crypto computations are done by local CPU instead of TPM for the reason discussed above. And also, RSA keys are chosen on purpose so that verification is much faster than signing, which favors the servers in computation. It takes less than 6ms to generate a RSA signature on the client computer, while on the server end it can validate more than 14000 RSA signatures per second.

Cases of client nodes from 1000 to 80000 are tested in the experiment, less than 5% of which are *good* clients. While *good* clients stick to the protocols faithfully, *evil* clients strike the DDoS attacks discussed in this paper. In every test, *good* clients are generated randomly on every client computer ranging from 2% to 5% of all the simulated clients on this machine. Every *good* client delivers 10 to 100 requests randomly in each test, while the *evil* clients attack the servers all through the test. The service latencies are obtained by *good* clients and then averaged to get the final results. All the programs are written in C++ using Linux socket programming.

Table I shows the latency increase (in ms) experienced by a normal client request in the old and new models. Normally, a user takes less than 1 ms to get one's service request fulfilled. Under DDoS attack, we find that the latency increases slowly in new models, and even an almost 80,000-node botnet can only degrade the performance of a normal request by 2.78 ms. While in old

TABLE I.
LATENCY INCREASE

Number of client nodes	New model	Old model
1000	0.01	672
5000	0.12	4863
10000	0.33	>15000
20000	0.65	/
30000	1.08	/
40000	1.34	/
50000	1.57	/
60000	1.98	/
70000	2.27	/
80000	2.78	/

models, the latency increases drastically to 672 ms at the beginning, and the DDoS attack quickly breaks the server down.

VI. CONCLUSION

Two new improved crypto puzzles are presented after an analysis and discussion of two existing crypto puzzles relying on the TPS with a trusted clock. And all the drawbacks found in the old ones are eliminated. The TPS in the new puzzles doesn't have to be equipped with a clock, thus can be realized by the TPM without modification. The new crypto puzzles can deal with some DoS attacks that the old ones can't. The new crypto puzzles protocols are simpler and more efficient than the old ones, while remains the same security. Prototype experiments also show that the new puzzle models can mitigate DDoS attacks well.

REFERENCES

[1] Q. Zhang, P. Wang, H. Yang. Applications of Text Clustering Based on Semantic Body for Chinese Spam Filtering.Journal of Computers.2012,Vol 7(11):2612-2616

[2] R. C. Merkle. Secure Communications Over Insecure Channels. Commun. ACM, 1978,Vol 21(4):294-299.

[3] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In CRYPTO, volume 740 of LNCS, Springer, 1992: 139-147

[4] M.Jakobsson and A. Juels. Proofs of Work and Bread Pudding Protocols. In CMS'99: Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks, 258-272

[5] T. Aura, P. Nikander, and J. Leiwo, DoS-Resistant Authentication with Client Puzzles, Proc. Security Protocols Workshop 2000, Springer-Verlag, New York, 2000: 170-177

[6] B. Waters, A. Juels, J. A. Halderman, and E.W.Felten. New Client Puzzle Outsourcing Techniques for Dos Resistance. In ACM Conference on Computer and Communications Security, ACM,2004:246-256

[7] R. Roman, J. Zhou, and J. Lopez. Protection Against Spam Using Pre-Challenges.In SEC, Springer, 2005: 281-294

[8] A.Back. Hashcash-a denial of service counter-measure. Technical report, 2002.

[9] D. Dean and A. Stubble_eld. Using Client Puzzles to Protect TLS. In SSYM'01:Proceedings of the 10th conference on USENIX Security Symposium, Berkeley, CA, USA, 2001. USENIX Association, 1-1.

[10] Yves Igor Jerschow, Modular Square Root Puzzles: Design of Non-Parallelizable and Non-Ineractive Client Puzzles,in Computers & Security volume 35. 2013:25-36.

[11] Q. Tang, A. Jeckmans, On Non-Parallelizable Deterministic Client PuzzleScheme with Batch Verication Modes, Centre for Telematics and InformationTechnology, University of Twente, http://doc.utwente.nl/69557/(2010).

[12] C. Dixon, T.Anderson, and A. Krishnamurthy. Phalanx: Withstanding multimillion-node botnets. Proc. NSDI,2008.45-58.

[13] B.Parno,D.Wendlandt, E.Shi,A.Perrig,B.Maggs,and Y.Hu. Portcullis:Protecting connection setup from denial-of-capability attacks. Proc. ACM SIGCOMM,2007: 289-300

[14] H. Wang,H. Zhou, C. Wang. Virtual Machine-based Intrusion Detection System Framework in Cloud Computing Environment.Journal of Computers.2012,Vol 7(10):2397-2403

[15] N. Borisov. Computational Puzzles as Sybil Defenses. In Peer-to-Peer Computing, IEEE Computer Society, 2006, 171-176

[16] Li F, Mittal P, Caesar M, et al. SybilControl: practical sybil defense with computational puzzles[C]//Proceedings of the seventh ACM workshop on Scalable trusted computing. ACM, 2012:67-78

[17] M. K. Franklin and D. Malkhi. Auditable Metering with Lightweight Security. In Financial Cryptography, volume 1318 of LNCS, Springer, 1997: 151-160.

[18] A. Serjantov and S. Lewis. Puzzles in P2P systems. In 8th Cabernet Radicals Workshop, Oct 2003.

[19] A. Juels and J. G. Brainard. Client Puzzles: A Cryptographic Countermeasure Against Connection Depletion Attacks. In NDSS. The Internet Society, 1999, 151-165.

[20] Douglas Stebila. Stronger difficulty notions for client puzzles and denial-of-service-resistant protocols,LNCS,volume 6558,2011:284-301.

[21] John Douceur. The Sybil Attack. In Proceedings of the 1st International Peer to Peer Systems Workshop, March 2002:251-260

[22] R.L.Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto. Technical Report MIT/LCS/TR-684,MIT,February 1996

[23] K. Chalkias and G. Stephanides. Timed Release Cryptography from Bilinear Pair-ings Using Hash Chains. In Communications and Multimedia Security, volume 4237 of LNCS, Springer, 2006: 130-140

[24] J. Cathalo, B. Libert, and J.-J. Quisquater. Efficient and Non-interactive Timed-Release Encryption. In ICICS, volume 3783 of LNCS, Springer,2005: 291-303

[25] Mahmoody M, Moran T, Vadhan S. Publicly verifiable proofs of sequential work[C]//Proceedings of the 4th conference on Innovations in Theoretical Computer Science. ACM, 2013:373-388.

[26] A.C.-F. Chan and I. F. Blake. Scalable, Server-Passive, User-Anonymous Timed Release Cryptography. In ICDCS, IEEE Computer Society,2005:504-513

[27] K. Chalkias, D. Hristu-Varsakelis, and G. Stephanides, Improved Anonymous Timed-Release Encryption, in ESORICS 2007: Proceedings of the 12th European Symposium On Research In Computer Security, Sep.2007:311-326

[28] J. A. Garay and M. Jakobsson. Timed Release of Standard Digtigal Signatures. In Financial Cryptography, volume 2357 of LNCS, Springer,2002:168-182
[29] D.Boneh and M. Naor. Timed Commitments. In CRYPTO, volume 1880 of LNCS, Springer,2000: 236-254
[30] Y.Dodis and D.H. Yum, Time Capsule Signature, in FC'05:Proceeding of the 9th International Conference on Financial Cryptography and Data Security,2005: 57-71
[31] Y. I. Jerschow, M. Mauve, Offline Submission with RSA Time-Lock Puzzles,in: CIT 2010: Proceedings of the 10th IEEE International Conference on Computer and Information Technology, 2010: 1058-1064
[32] TPM Work Group. TCG TPM Speci_cation Version 1.2 Revision 103. Technical report, Trusted Computing Group, 2007.
[33] P. Tsang and S. Smith, Combating spam and denial-of-service attacks with trusted puzzle solvers, in Proceedings of the Information Security Practice and Experience Conference, 2008:188-202
[34] C. Liu, D. Liu.QoS-oriented Web Service Framework by Mixed Programming Techniques. Journal of Computers.2013,Vol 8(7):1763-1770
[35] G. Karame and S. Capkun, Efficient Client Puzzles based on Repeated-Squaring. ;In Proceedings of IACR Cryptology ePrint Archive. 2009:607-607