Apply Quantum Search to the Safety Check for Mono Operational Attribute Based Protection Systems

Vincent C. Hu

Computer Security Division
National Institute of Standards and Technology
Gaithersburg, U.S.A. 20899
vhu@nist.gov

Abstract. Interrelated computing device's system such as IoT, RFID, or edge device's systems are pervasively equipped for today's information application and service systems, protecting them from unauthorized access i.e. safety is critical, because a breach from the device may cause cascading effects resulting to data lost or even crash of the whole information system. However, to determine a protection system's safety is proven to be undecidable unless the system has limited management capabilities. And even with such limitation, it is too expensive to perform a safety test in term of computation time when a device has more than hundreds of subjects which is not uncommon for interrelated computing devices. Nevertheless, the required exponential computing time for safety test can be significantly reduced to its square root if computed by quantum algorithm. In this paper we demonstrate an application of quantum search algorithm to reduce the computation time for safety test for limited (i.e. mono operational) protection systems which are based on attribute-based access control model. The improvement of the performance allows the safety test for interrelated computing device's system to be much less expensive to compute.

Keywords: Access Control; Protection System; Quantum Algorithm; Security Model; Quantum Search.

1 Introduction

Interrelated computing device's systems (ICDSs) [1] such as IoT, RFID, edge device's or similar systems are pervasively equipped for today's information applications and services [2], protecting these systems from unauthorized access i.e. safety is critical, because a breach from an ICDS may cause cascading effects leading to data lost or even crash of the whole system[3]. Further, as Attributed-Based Access Control (ABAC) [4] model is getting more applied for access control [5], an ICDS may apply ABAC for its access control mechanism. We call such protecting system the **Attribute Based Protection System** (ABPS), which also includes its access control **policy management functions**.

The **safety** for an ABPS is to ensure that it is impossible to leak access **privilege** (perform actions to objects) from authorized subjects to unauthorized subjects through any changes of access state. And **safety test** is to verify if the safety of the system is maintained after any order of access control policy changes. To test that in worst case obviously requires checking access control policy updates evoked by all possible sequences of policy management functions. By HRU[1][6] theory, such test is proven to be undecidable unless the ABPS is limited to be **mono operational**, which is restricted to have only one primitive commend for each policy management function.

An ICDS's ABPS can be mono operational, because its access control requires limited management capability. But even that, the exponential computation time (NP-Complete) for safety test is still too expensive [7], because subjects, objects, and actions as exponential variables of computing time for most ICDSs can easily reach to hundreds if not thousands. Nevertheless, it can be significantly improved by applying quantum search algorithm, which reduces the computation time to the square root of the time required by classical algorithm, thus, allows the safety test for ICDS's ABPS to be minimum computable.

This paper is divided into six sections, section I is the introduction, Section II describes the ABPS, Section III explains the safety test algorithm that applied to mono operational ABPS, Section IV introduces quantum algorithm modified from quantum search algorithm for the privilege leak detect process of safety test algorithm. Section V demonstrates the performance comparison between quantum and classical algorithms in terms of computation time, and Section VI is the conclusion.

2 Attribute Based Protection System

Attribute Based Access Control (ABAC) is an access control method where subject requests to perform actions on objects are granted or denied based on assigned attributes of the subject and objects, environment conditions, and a set of rules specified by those attributes and conditions [4] called ABAC **policy**, which given the values of the attributes of the subject, object, and environment conditions and their relations make it possible to determine if a requested access should be authorized.

ABPS applies ABAC where a **subject** s_i represents a combination of **subject attributes** $sa_1,\ldots sa_i,...sa_k$ the subject is associated with, and an **object** o_i represents a combination of **object attributes** $oa_1,\ldots oa_j,...oa_l$ that apply to the object. And the access control policy is managed by the policy management functions. The ABPS's access state can be presented by the **HRU access matrix** (Figure 1) such that the access control policy rules are mapped to rows and columns with intersected cells. A cell contains actions that are permitted to perform the accesses from the subject to the object corresponding to the row and the column, as example in Figure 1, shows that subject s_j is permitted to perform actions *r* and *w* accesses to object o_i, The cell intersected by both row and column of subjects is used for creating or deleting a subject by another subject. ABPS's ABAC policy rules

[1] We denote the term HRU to be general references to the systems and theories presented in [5].

are mapped to the access matrix by adding permitted access actions into cells and removing denied accesses actions from cells if the actions existed.

Subject\Object	s_1	…	s_n	o_1= (….)	…	o_i = (oa_1,…oa_j,…oa_l)	…	o_m= (….)
s_1 = (….)								
…..								
s_j = (sa_1,..sa_i,..sa_k)						*r, w*		
…..								
s_n = (….)								

Fig. 1. ABPS access matrix state.

An ABPS's policy management mechanism, which in general is a set of policy management functions for creating, updating the ABAC policy rules. The function is intrinsically equal to access **matrix update function** such that *assign rule* function: "*assign action a to object o_j to subject s_k*" is equal to: "*add a to the intersect cell of row s_k and column o_j*" add function, and *delete rule* function: "*delete action a to object o_j of subject s_k*" is equal to: "*remove a in the intersect cell of the row s_k and column o_j*" delete function. Figure 2 shows an example ABAC rule: "*users with attribute p or q can read device x*" maps to HRU access matrix. Therefore, an ABPS access state is an instance of an HRU access matrix state, and ABAC policy rules can be configured to rows and columns of an HRU access matrix.

Users/Device	…..	device *x*	
…….			
Attribute *p*		*read*	
…….			
Attribute *q*		*read*	

Fig. 2. ABPS access matrix state

There are six **primitive commands** for ABPS's policy management functions, and their counter parts for access matrix operations are shown in Table 1:

Table 1. ABPS'S and HRU primitive commands mapping.

ABPS primitive commands	HRU primitive commands
assign action (a, s_i = (sa_1…..sa_k), o_i = (oa_1…..oa_j))	*enter action a into* (s_i, o_i)
delete action (a, s_i = (sa_1…..sa_k), o_i = (oa_1…..oa_k))	*delete action a into* (s_i, o_i)
add subject (s_i = (sa_1…..sa_k))	*create subject* s_i
add object (o_i = (oa_1…..oa_k))	*create object* o_i
remove subject (s_i = (sa_1…..sa_k))	*destroy subject* s_i
remove object (o_i = (oa_1…..oa_k))	*destroy object* o_i

Access state changes after executing sequences of access state change functions can be presented formally by: $Q_1 \vdash fn_1\ Q_2 \vdash \ldots\ fn_i\ Q_i \vdash \ldots\ fn_m\ Q_m$, ($\vdash$ means complete the function) where Q_i is an access state, and function fn_i makes access state change from Q_i to Q_i+1. The pseudo fn_i for ABPS policy management function is:

ABPS_fn$_i$ (*subjects*, *actions*, *objects*){//* *subjects* or *objects* are optional if the functions are *create*/*destroy* subjects or objects *//
if no conflict with current ABAC access control policy
then { execute primitive commands pc_1;
execute primitive commands pc_2;
….
execute primitive commands pc_n,
that apply to the *subjects*, *actions*, and *objects*
}//*primitive commands update ABAC policy *//
current access control policy $P_i = P_{i+1}$
}

And the corresponding fn_i for HRU access matrix update function is:

HRU_fn$_i$ (*subjects*, *actions*, *objects*) { //* *subjects* or *objects* are optional if the functions are *create*/*destroy* subjects or objects *//
if conditions c_1, c_2….c_k then {
execute primitive commands pc_1;,
execute primitive commands pc_2;,
….
execute primitive commands pc_n,
that apply to the *subjects*, *actions*, and *objects*
} //*primitive commands update access matrix *//
curent access matrix $H_i = H_{i+1}$
}

The steps for checking the conflict of access control policy in *ABPS_fn*$_i$ are not semantically different from HRU_*fn*$_i$'s *if* condition checks, because satisfying the current ABPS policy is the same as satisfying the state of HRU matrix, which can be translated from the ABPS's policy rules.

3 ABPS safety check

HRU defines that:

"*given a protection system, we say command* ***c*** *leaks generic action* ***a*** *from the access state if* ***c***, *when run on the access state, can execute a primitive operation which enter* ***a*** *into a cell of access matrix which did not previously contain* ***a***". from the definition, the **safety** of ABPS is to ensure unintended subjects cannot perform protected actions on objects through executing any sequence of policy management functions (*ABPS_fn*$_i$s described in Section II), such that the permitted accesses for the action by the original access control policy remains the same

during the system's life cycle. Thus, the **safety test** is to verify that if the system remains safe after all possible sequence of policy management functions being executed. Figure 3 shows the components and relations of an ABPS and its safety test system.

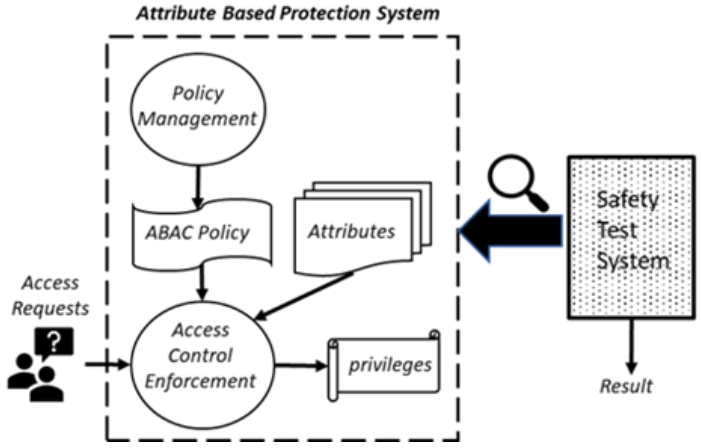

Fig. 3. ABPS and Safety test system

According to HRU, safety test for general protection systems including ABPSs is undecidable in term of computation complexity, because to test the safety of a protection system in worst case obviously requires checking all possible sequences of access matrix changes evoked by access matrix update functions with all possible parameters including subjects, actions, and objects, plus that the function may contain unlimited *if* condition checks and arbitrary numbers of primitive commands (Table 1). The undecidability can be proved by configuring the protection system to simulate the behavior of an arbitrary Turing machine, with a safety leakage state corresponding to the Turing machine entering a final state [6].

In addition to general protection systems, from HRU's theory, a restricted type of protection system called **mono operational protection system**, which limits each matrix update function to contain only one primitive command. HRU shows that determining safety for mono operational protection system is decidable in NP-Complete, which is proved by reducing a K-clique problem to a safety decision problem that translating the system's initial access matrix to an **adjacency matrix** for a graph, then test to see if it forms a *k*-clique before entering an action *a* to the access matrix causing safety leak. HRU also shows that only the primitive command *enter* can change the access state. To simulates the HRU's algorithm for mono operational ABPS safety test, Figure 4 illustrates the algorithm *Safety_Test* for an action *a*. Since a policy management function of a mono operational protection system contains only one primitive command, and only the *enter* primitive command can change the access state, the algorithm needs to tests every possible sequences of *enter* commands for all actions, in other words, try all possible sequences of primitive *enter* commands, (optional starting with a *create* subject command) of length up the $|A|\times|S|\times|O|$ for each sequence, where $|A|$ is the number of all actions, $|S|$ is the number of all subjects, and $|O|$ is the number of all objects.

The parameters of an *enter* command are an action-subject-object triplet corresponding to a command sequence, which is identified by a binary number, for example, if there are two subjects s_1 and s_2, two objects o_1 and o_2 , and two actions a_1 and a_2 in the ABPS then there are 2×2×2 = 8 different *enter* command, and 2^8 possible sequences, for instance, the 5th *enter* command sequence is {*enter* (a_1, s_1, o_1); *enter* (a_1, s_2, o_1)}, and the 24th command sequence is {*enter* (a_1, s_1, o_1); *enter* (a_1, s_2, o_2); *enter* (a_2, s_1, o_1)}, because the binary form of the sequence 5 is 00000101 and 24 is 00011001, where the bits representations of *enter* commands are assigned in Table 2.

Table 2. Example bit number assignment of 2 actions, 2 subjects and 2 objects pairs.

Triplet	*Assigned bit*
(a_1, s_1, o_1)	1st bit
(a_1, s_1, o_2)	2nd bit
(a_1, s_2, o_1)	3rd bit
(a_1, s_2, o_2)	4th bit
(a_2, s_1, o_1)	5th bit
(a_2, s_1, o_2)	6th bit
(a_2, s_2, o_1)	7th bit
(a_2, s_2, o_2)	8th bit

Safety_Test (P_1, a) { (1)
H_1 = *Initial* P_1; //* map accesses permitted by the ABPS access control policy to the HRU access matrix *// (2)
Privilege_leak = 0 ; (3)
i = 1; (4)
For k = 0 to $2^{|G|\times|S|\times|O|}$ - 1 //* $|G|$ is the number of actions, $|S|$ is the number of subjects, $|O|$ is the number of objects *//{ (5)
For all (a_x, s_i, o_j) //*(a_x, s_i, o_j) ∈ {$A\times S \times O$}; A is the set of actions, S is the set of subjects, O is the set of objects *// (6)
{
If *Bitmap* (a_x, s_i, o_j, k) = 1 //*match s_i-o_j pair to binary number k*// (7)
enter (a_x, s_i, o_j) (8)
{
}
$H_i = H_{i+1}$; (9)
If *State_Compare* (H_i, H_1, a) //*check if access state is changed*// (10)
Then { (11)
privilege_leak = 1; (12)
end *Safety_Test*; (13)
}
else *privilege_leak* = 0; (14)

```
    }
}
Bitmap (a_x, s_i, o_j, k) {
        i = Numer_map(a_x, s_i, o_j) //* translate s_i-o_j pair to binary number*//
        For j = 1 to i
        If the jth bit of Binary(k) == 1 //*check the match of bits*//
                return 1
}
State_Compare (H_i, H_1, a){
    For each row of s_i {
        For each column of o_i {
            If (((a in the cell of (s_i, o_i)) of H_i) == ((a in the cell of (s_i, o_i)) of H_1))
                Then return leak = 0
                else return leak = 1; //*privilege_leak state is passed to Safety_Test*//
        }
    }
}
```

Fig. 4. ABPS Safety Test algorithm

The *Bitmap* function translates the action-subject-object triplet of the *enter* command to a binary number to match the current sequence number passed to the function as examples showed in Table 2.

The *State_Compare* function compares cells in original access matrix H_1 to the new access matrix H_i that might be updated after a sequence of *enter* commands were executed, it checks if a privilege leak by action *a* is found, and the result is returned to the *Safety_Test*. Note that the algorithm only checks the safety against one action *a*, it is capable of checking multiple actions leaks, and to do that we need to replace (s_i, o_j) with (s_i, a_m, o_j), $\{S\times O\}$ with $\{S\times A\times O\}$, $2^{|S|\times|O|}$ with $2^{|S|\times|A|\times|O|}$, and (a, s_i, o_j) with (a_m, s_i, o_j) and add a For loop for each a_m check in the function.

For later discussion of quantum algorithm, we call the *For* loop from line 5 to 14 in Figure 4 the *Leak_Detect* process collectively. Hence, the *Safety_Test* would require $2^{|A|\times||S|\times|O|}\times$ O(*Leak_Detect*) computation time (steps) for detecting an access privilege leak, where O(*Leak_Detect*) is the time needed for *Leak_Detect* process, which is equal to O(*Bitmap*)$\times$ $|A|\times|S|\times|O|$+O(*State_Compare*) = $2\times|A|\times|S|\times|O|$, because O(*Bitmap*) take constant and O(*State_Compare*) takes number of steps equals to the size of access matrix: $|S|\times|O|$ times $|A|$ to compute.

Some low power ICDSs' (e.g. IoT, RFID, or edge computing devices or similar systems) access control are managed by ABPS, where access control policies are either embed or deployed by central management system rather than managed by the device themselves [8]. For instance, RFID devices include independent storage access control rules, only when the rule needs to be updated, do reading devices need to communicate with the server, and access control rules can be updated by the multicast method. In the same security zone, multiple reading devices can distribute access control rules at the same time, thereby improving the efficiency of rule updates [9]. In addition, some access control mechanisms allow smart objects take the authorization decisions based on current context of the processes in

use [10]. For those systems with limited access control management capabilities, the protection systems can be implemented by mono operational ABPS. And these ICDSs usually accessed by a large number of users risking safety leak [11], plus, due to frequently adding new and updating old devices, their safety need to be efficiently verified to satisfy their security and performance requirements of services, thus, need an efficient safety test method that classical algorithms cannot offer.

4 Quantum search algorithm for ABPS safety check

Even the ABPS safety test is decidable but in NP-Complete as described in the last section, it is still an issue to be efficiently computable for systems having large number of subjects, objects and actions such might sum up to hundreds if not thousands of users, because, for example, an ICDS is used by just 10 subjects (classified by users' attributes) with only two objects (classified by devices' attributes) and 3 actions, the safety will take $2^{10\times2\times3} \times (2\times10\times2\times3)$ computation time. Thus, it is desirable to improve the exponential computation time (steps) to be feasible to compute. To reduce the computation time, we propose to adopt the Grover Quantum search algorithm [12, 13], which performs the transformation $L|x\rangle|q\rangle = |x\rangle|q\otimes f(x)\rangle$ to a black box oracle f to speed up $f(x)$ for multiple x inputs, where $|q\rangle$ is an ancilla qubit for quantum unitary computation. The algorithm finds with high probability the unique input to the black box oracle function that produces a particular output value, using just $\sqrt{N}$ evaluations of the function, where N is the size of the function's domain.

Schema in Figure 5 shows the application of quantum search algorithm for safety test called ***Safety_Test* quantum algorithm**, which uses n+1 qubit register as input (the ancilla 1 qubit is for quantum unitary operation), where $N = 2^n = 2^{|A|\times|S|\times|O|}$ is the number of all possible sequences of *enter* commands, $|A|$ is the number of actions, $|S|$ is the number of subjects, and $|O|$ is the number of objects of the ABPS. The output of the algorithm is a number x_{leak} representing a sequence of *enter* commands that causes privilege leak by the action a. Notice that instead of a leak command sequence, the classical *Safe_Test* algorithm (Figure 4) only returns a result indicating whether a leak exist. In contrast, the quantum algorithm returns one of the leak sequence numbers (there could be more than one command sequence that cause leakages). The black box oracle function f is hence the *Leak_Detect* process (from line 5 to 14 of the classical *Safety_Test* algorithm in Figure 4).

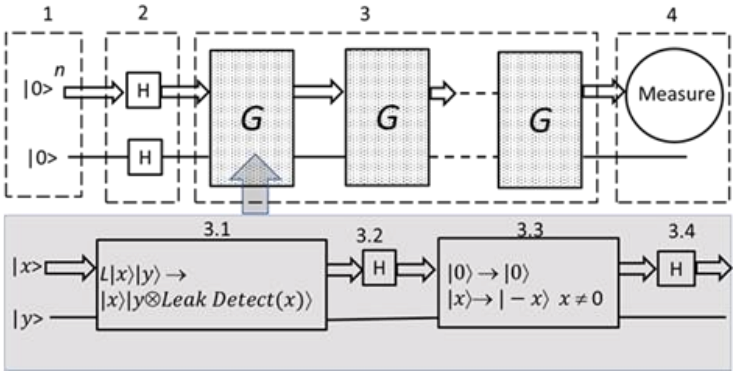

Fig. 5. Quantum Safety Check schema

Safety_Test quantum algorithm requires repeating applications of the Grover quantum search subroutine shown as the **Grover iteration** *G* in Figure 5, where each iteration move $1/\sqrt{N}$ amplitude towards solutions, thus $\sqrt{N}$ iterations should suffice to render a x_{leak}. The algorithm is divided into four steps as below

1) Begins with the initial state, $n + 1$ qubits in the state $|0\rangle:|0\rangle^{\otimes n}|0\rangle$, the extra $|0\rangle$ is for the quantum unitary operation.

2) The Hadamard transform is applied to establish equal superposition state $|\Psi\rangle$ of all possible numbers of *enter* command sequences that

$$|\Psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

Where $0 \le x \le 2^{|A|\times|S|\times|O}$.

3) Apply the Grover *G* iteration $K=\lceil \sqrt{N/M} \rceil$ times: where *M* is the number of sequences of *enter* command sequences (i.e. x_{leak}s) that cause privilege leaks. This step can be subdivided into the following three steps:

3.1) Apply the quantum oracle *L*

$$L|x\rangle|y\rangle = |x\rangle|y \otimes Leak\ Detect(x)\rangle$$

resulting

$$|x\rangle \to (-1)^{Leak\ Detect(x)}|x\rangle$$

Note that each *x* is a number representing an enter commands sequence, for example the number 5 represent the sequence; { *enter* (g_2, s_1, o_2); *enter* (g_1, s_2, o_1)} as shown in Section III. *Leak_Detect* (*x*) = 0 for all $0 \le x \le 2^n$ except the x_{leak} for which *Leak_Detect*(*x*) = 1 indicating the *enter* command sequence leaks privilege for action *a* in the current access control state.

3.2) Apply the Hadamard transform $H^{\otimes n}$

$$|\Psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} (-1)^{Leak\ Dectect(x)} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

3.3) Performs a conditional phase shift i.e.

$$|0\rangle \to |0\rangle \text{ and}$$
$$|x\rangle \to |-x\rangle, \ x \neq 0$$

with every computational basis state except $|0\rangle$, receiving a phase shift of -1, i.e. the leaking *enter* command sequence $x = x_{leak} \neq 0$. The conditional phase shift can be calculated by applying the matrix operation of

$$2|0\rangle\langle 0| - I$$

where I is the identity matrix.

3.4) Apply the Hadamard transform $H^{\otimes n}$

4) Measure the first n qubits of $|\phi\rangle$ gets one of the possible leak sequence x_{leak}.

The quantum algorithm requires $\sqrt{N/M} \times$ O(*Leak_Detect*) [14] of computation time, where M is the number of *enter* command sequences that cause leaks, in other words, there could be multiple leaking *x*s, so, M implies that there is at least one leak sequence exist. After the Grover iterations (calls to oracle *Leak_Detect*) were performed, one of the M sequences will be measured out with higher probability than the sequences that may or may not causing leak. The algorithm is a quadratic improvement over the $N/M \times$ O(*Leak_Detect*) calls performed by classical computer.

Since measuring from step (4) will render only one result, however, there could be cases that has no or multiple leak sequences exist, hence the result could be a random sequence i.e. mistakenly identified as a leaking sequence. To correct this inaccuracy, three methods can be applied:

a) A planned fake leak sequence x_f: *enter* (g_f, s_f, o_f) is assigned in between line 9 and 10 of *Leak_Detect* process in the *Safet_Test* algorithm such that the x_f will be detected *as* a leak command sequence in $\sqrt{N}$ time with high probability close to 100%, because it is the only leak sequence can be detected that makes $M = 1$. And if after several runs of the algorithm, the results repeatedly measured to be the same x_f, we can confidently determine that there are no other leak sequences besides the planned x_f.

b) To more precisely determine the number of leak command sequences, combine the Grover iteration G with **quantum counting** algorithm [14]. The method is to estimate the number of leak command sequences by quantum counting, which is an application of the phase estimation procedure to estimate the eigenvaluses $e^{i\theta}$ of Grover iteration G, which in turn enables determining an approximate number of leak command sequences M. The method allows us to decide whether a leak sequence even exists depending on the result number. The phase estimation circuit used for quantum counting is shown in Figure 6. The function of the circuit is to estimate θ to an accuracy approximate to 2^{-m} (note[2]).

[2] More accurate, m should be $m+\lceil \log(2+1/2\epsilon) \rceil$ qubits .

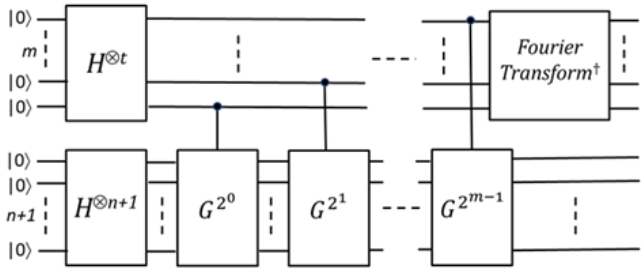

Fig. 6. Approximate quantum counting circuit for *G*

c) requires no additional process, but repeatedly running the algorithm enough times, then analyze the measured results. If there is no command sequence causing leak, any random sequence number will be measured with the same probabilities of all other sequence numbers. Such result indicates that there is no concentrate output of one particular leak sequence number meaning that the possibility of having a true leak sequence is low, however, this method is reliable only when the total number of actions, subjects and objects is large enough for the odd that getting a random result, which is true leak sequence is low. Table 3 compares the three methods.

Table 3. Comparison of testing methods for checking the existence of true leak sequences.

Checking methods	If true leak sequences exist	It no true leak sequence exist	Accuracy
(a) Plan fake leak sequence access x_f	Equal probabilities of getting true leak sequences x_{leak} and x_f	Fake leak sequence x_f has highest possibility being measured	Median
(b) Quantum counting by applying phase estimation	Number of solutions from phase estimation algorithm > 0	Number of solutions from phase estimation of algorithm ≈ 0	Hight
(c) No extra step required but run the algorithm enough times	High probability of a true leak x_{leak} sequence is measured	Equal probability for every sequence will be measured	Low (reliability increased by increasing the number of input (i.e. subjects and objects) sequences)

Table 3 shows that the more difficult in implementing the method (as ordered by methods *b*, *a*, and *c*) the more accurate result it will generate, unless depending on the number of possible sequences in method *c*, which if applied to a large number of total sequences (say no less than hundreds) then the accuracy might equal or better than method *a* and *b*, however, repeating the process of *c* method is not as efficient as the other methods. The detail algorithms and comparison of the three methods is interesting that worth to be discussed by their own topics, due to the limited space and to keep the discussion on focus, we only briefly introduce them in this paper.

5 Performance of safety check quantum algorithm

We can now summarize the performance improvement of safety test for a mono operational ABPS by comparing quantum to the classical algorithms. Assuming there is at least one leak sequence exist, by the quantum safety test algorithm, it will take $\sqrt{N} \times$ O (*Leak Detect*) while classical safety test algorithm requires $N \times$ O (*Leak_Detect*) (for simplicity of demonstration, let's assume $M = 1$). The difference is in the order of $\sqrt{N}$ compared to $N = 2^{|A|\times|S|\times|O|}$, which is the 2's power of the number of actions |A| times the number of subjects |*S*| times the number of objects |*O*| managed by the ABPS system. For ICDSs or applications accessed by large number of subjects (users classified by attributes) to multiple number objects (devices classified by attributes), the quadratic difference is significant as shown in comparison listed in Table 4. Note that for the purpose of comparison, the O (*Leak_Detect*) is not counted, because both algorithms take the same polynomial time which does not affect the exponential difference.

Table 4. Computation time comparison of classical and quantum algorithms for ABPS safety test

Number of subjects times objects	Classical algorithm × O (*Leak_Detect*)	Quantum algorithm × O (*Leak_Detect*)
5	32	5.6568542494492 ≈ 6
10	1024	32
15	32768	181.0193359838 ≈ 181
20	1048576	1024
25	33554432	5792.61875148 ≈ 5793
30	1.073741824×10^{9}	32768
35	$3.4359738368\times10^{10}$	185363.8000474 ≈ 185364
40	$1.099511627776\times10^{12}$	1048576
45	$3.518437208883\times10^{13}$	5931641.601516 ≈ 5931642
50	$1.125899906843\times10^{15}$	33554432

The growth of computation time from 5 to 50 (number of subjects times objects) is about 3.5×10^{13} time for classical algorithm, and about 6×10^{6} time for quantum algorithm, obviously, the improvement of quadratic reduction by quantum algorithm allows the safety test to be reasonably performed.

An ICDS's device in general is accessed by only one public user class (subject with *public* attribute) plus one administrator (subject with *administrator* attribute) and limited actions available to manage the device, so, at minimum, two subjects can read and write (most common actions) to the object, thus, only require $2^{2\times2\times1}\times$ O (*Leak_Detect*) computation steps by classical algorithm for safety test. However, some ICDSs may have more than one device to be managed, so the access control policy is deployed from central service to individual device as described in Section III. In such cases, the ICDS's ABPS may apply one-size-fits-all access control policy to its devices, thus even with limited allowed actions, but has multiple

number of administration subjects and device objects. Further even with a single device (object), it is not uncommon that an ICDS has more than tens even hundreds of subjects and objects. So the computation time for safety test is not practical by classical algorithm for these systems, but if instead use quantum algorithm, the difference is enormous even for small number of actions, subjects and objects as shown in Figure 7, the growth is measured in 1000 computation time per unit for up to 20 in comparison of classical and quantum algorithm. It shows that there is not much benefit using quantum algorithms if the number is less than 10, however, the difference is obvious when the number is greater. Note that the comparison is for detecting leak for only one action, if there are multiple actions involved, the computation time will increase even exponentially greater by $|A|$, which is the number of the actions under test as a factor of exponent.

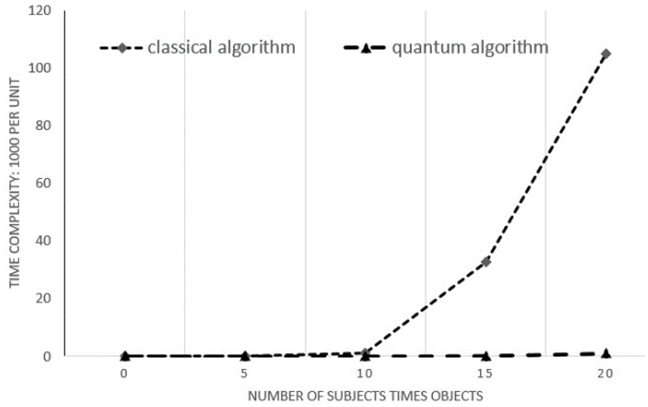

Fig. 7. Computation time comparison of quantum and classical algorithms for safety test

6 Conclusion

To determine the safety of a protection system is to find if there are privilege leaks from protected actions to unauthorized subjects of the system. HRU shows that for mono operational protection system, the computation time for the safety test is decidable, however take NP complete computation time, which is too expensive to perform for a system with large number of subjects and objects such as ICDS (e.g. IoT, RFID systems etc.) that applies attribute based access control (ABAC) model.

We demonstrate that an ABPS (protection system that applies ABAC model) such as ICDS can be simulated by an HRU access matrix and its matrix management functions. And adapted from Grover quantum search algorithm, we propose a quantum safety test algorithm, which determines the safety by returning a command sequence that will cause access leak for a mono operational ABPS. We conclude that if N equals to $2^{|A|\times|S|\times|O|}$ where $|A|$ is the number of actions, $|S|$ is the number of subjects, and $|O|$ is the number of objects, and each of subject, object

represent a set of attributes associate to them, the quantum algorithm for a mono operational ABPS requires computation steps $\sqrt{N}$ times the time required for classical leak detection process, compared to *N* times the time required for classical leak detection process, the quantum algorithm reduces the computation time quadratically. The saving is significant for ICDS or similar systems that its devices usually are accessed by large number of subjects with limited available actions. In addition to the quantum algorithm, three methods are explained to ensure that the test result are genuine instead of some random command sequence that does not but mistakenly rendered as a command sequence that causes access privilege leak.

References

1. Rouse, M.: Internet of Things (IOT), IoT Agenda, Tech Target, https://internetofthingsagenda.techtarget.com/definition/Internet-of-Things-IoT (2019).
2. Voas, J., Kuhn, R., Laplante, P., Applebaum, S.: Internet of Things (IoT) Trust Concerns. NIST Cybersecurity White Paper (2018).
3. Siboni, S., Glezer, C., Shabtai, A., Elovici, Y.: A Weighted Risk Score Model for IoT Devices. In: SpaCCS 2019 International Workshops proceedings, pp 20-34. Springer, Atlanta. GA, USA (2019).
4. Hu. V. et. al: Guide to Attribute Based Access Control Definition and Considerations", National Institute Standards and Technology", NIST SP 800-162. (2014).
5. AXIOMATIC: Attribute Based Access Control (ABAC), https://www.axiomatics.com/attribute-based-access-control/.
6. Harrison, M. A., Ruzzo, W. L., Ullman, J. K.: Protection in Operating System, Communications of the ACM Magazine, Volume 19 Issue 8, pp 461-471 (1976).
7. Xu, Z., Li, X.: Secure Transfer Protocol Between App and Device of Internet of Things. In SpaCCS 2017 International Workshops, Proceedings, pp 25-34, Guangzhou, China (2017).
8. Skarmeta, A. F., Hern´andez-Ramos, J. L., Moreno, M. V.: A decentralized approach for Security and Privacy Challenges in Internet of Things, IEEE World Forum on Internet of Things, https://ieeexplore.ieee.org/abstract/document/6803122 (2014)
9. Dhillon, P., Singh, M.: Internet of Things Attacks and Countermeasure Access Control Techniques: A Review, International Journal of Applied Engineering Research ISSN 0973-4562 Volume 14, Number 7 pp. 1689-1698 © Research India Publications. http://www.ripublication.com (2019).
10. Mali, A., Darade, S.: Security and Privacy in Web-based Access Control in Internet of Things, Academia, https://www.academia.edu/28002646/Security_and_Privacy_in_Web-based_Access_Control_in_Internet_of_Things.
11. Maddison, J.: The Importance of Access Control for IoT Devices", SECURITYWEEK, https://www.securityweek.com/importance-access-control-iot-devices (2018).
12. Grover, L.: A fast quantum mechanical algorithm for database search. In Annual ACM Symposium on the Theory of Computation, page 212-219, ACM Press, New York (1996).
13. Grover, L. K.: Quantum mechanics helps in searching for a needle in a haystack, Phys. Rev. Lett, 79(2):325, 1997 arXive e-print quant-ph/9706033 (1997).
14. Nielsen, M., Chuang, I. L.: Quantum Computation and Quantum Information, Cambridge University Press (2000).